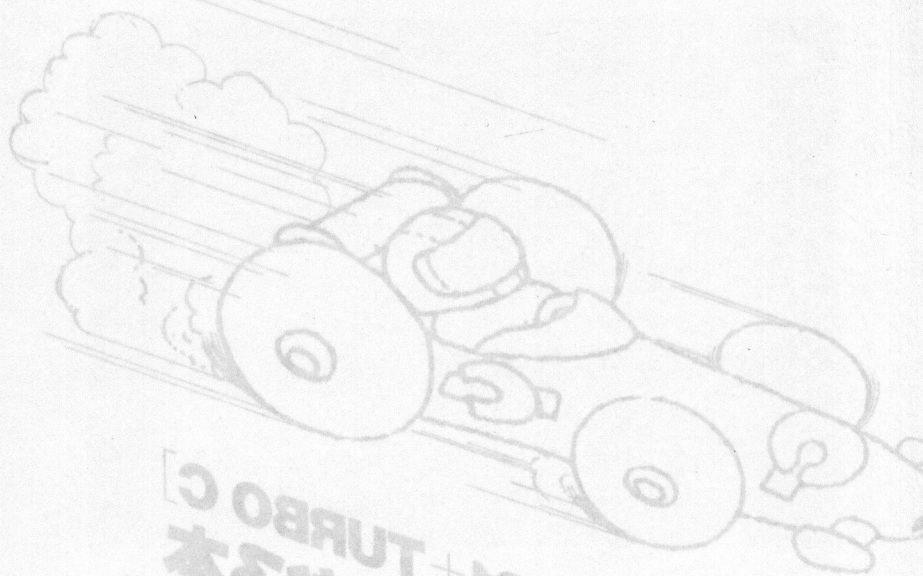
[PC-9801+TURBO C]
グラフィックスに強くなる本
ゲーム編
●青山 学 著
啓学出版

[PC-9801＋TURBO C]
グラフィックスに強くなる本
ゲーム編
●青山 学 著
啓学出版

第 4 章で使用するパターンデータ

第 3 章のパターンエディタで作ります。

「PC-9801+TURBO C」グラフィックスに強くなる本

ゲーム編

●青山学著

啓学出版

はじめに

コンパイラというと、ほんの少し前までは、パーソナルに所有することは夢物語として語られるだけでした。しかしながら、現在ではTURBO Cのような統合環境下で、プログラムの編集、実行、デバックができる手軽なコンパイラの登場により、様相は一変してしまいました。しかも、MS-DOSが普及するにつれ、マシンを自在に使いこなしたいという欲求を実現するための有力な手段として、C言語は不動の地位を占めてきたようです。

実際、C言語は様々な欲求に答えてくれます。データベースやエディタだけではなく、コンパラやOSまでもC言語で記述されるに至っては、C言語のスーパースターぶりを物語っていると言えるのかもしれません。

さて、本書ではグラフィックスの中でも人気の高い分野であるTVゲームをテーマとして掲げています。今までこの分野は、難解なマシン語でしか開発できない特殊な分野とされてきましたが、高速な処理を要求されるリアルタイムゲームでさえも、工夫次第ではC言語で作ることが可能となってきました。もちろんゲームでは、ハードウェアに強く依存したプログラムとなるのは避けられないことですから、対象となる機種はPC-9801シリーズに限ることになります。しかしながら、逆に考えれば、機種に強く依存したプログラムも記述できるというのがC言語の大きな特徴であるともいえるのです。

本書では、2進数の基礎から始まり、必要とされるハードウェアの知識を解説すると共に、C言語の初心者でも読めるように各関数は簡単なものばかりを用意しました。そして、それぞれの関数は機能別に部品化を図ったものとし、これらをプロジェクトファイルとして組み上げていくことに

よって、お絵かきツール、パターンエディタ、マップエディタ、そして最終的にはリアルタイムゲームを完成するという構成になっています。本書を通してTURBO Cの新たなる可能性を開拓する楽しさを、ぜひ味わってください。

なお、本書では、次のようなシステムを想定しています。

1 …… PC-9801シリーズ本体

2 …… ディスプレイ（640×400ドットのカラーディスプレイ）

3 …… MS-DOSシステム

4 …… TURBO C Version 2.0以降

また、PC-9801 VM21以降の機種では640Kバイトのメモリを標準で装備していますから問題ないのですが、もしも、640Kバイトのメモリが装備されていない機種であれば、サンプルプログラムの中でコンパイルできないものもありますから注意して下さい。

最後になりましたが、本書の執筆に際し、いろいろとご協力くださった、啓学出版の嶋貫健司氏、パソコンクラブA.Gプロジェクトの大野真也、佐藤新市、柴田錦見、清水明、寺内一郎の各氏と、処理系を提供していただいた(株)マイクロソフトウェア アソシエイツの方々に、この場より改めて御礼申し上げます。

1991年7月7日　　青山　学

もくじ

第1章　0と1だけの世界

現在ではコンピュータの技術もICやLSI、VLSIの時代になりましたが、設計された当初は、外部からのノイズや素子のばらつきなどから信頼性が問題でした。そこでコンピュータは、電流が有るか無いか電圧が正か負かなどの単純な2つの状態を基準として動作するように作られたのです。

たった2つの状態ですが、これだけでも片方を0もう一方を1とすることによって、あらゆる数字を2進数で表現できるわけです。そして、これら2進数の数値に、加算せよとか引き算せよなどの特別な意味を持たせたマシン語コードといわれるもので、命令が組み立てられたのです。

いまではその当時と比べると格段に技術も進歩したのですが、やはり現在でも多くのコンピュータが同じ仕組みで動いています。

ところで、コンピュータが2進数の数値命令で動いているといっても、人間が直接2進数の数値命令を出しているわけではありません。通常は高級言語と言われる命令でプログラムを記述し、そのプログラムをコンピュータ自身にマシン語コードへと翻訳させているのです。このように、プログラムはどんな言語で書かれたものでも最終的には、マシン語コードに翻訳してから実行されます。なお、マシン語に翻訳する前のプログラムをソースプログラムといいます。

本書で扱っているC言語も例外ではなく、C言語で書かれたソースプログラムをなんらかの翻訳者を通して、マシン語へと翻訳したものを実行することになります。この翻訳者の形態は、インタープリタ型とコンパイラ型の2つに大別されます。

インタープリタは、ある言語で書かれたソースプログラムをひとつひとつ読み込んでは、その文字列が何をすべきかを解釈して、あらかじめ組んであったマシン語のプログラムの中から、対応している処理を実行する。というタイプにつけられた総称です。

この方法では、インタープリタ本体と高級言語で書かれたソースプログラムだけですみますから、メモリは少なくてすみます。ただし、翻訳しながら実行するという過程では、実行時に翻訳時間が加わることになりますから、その分スピードはどうしても遅くなります。

現在のパソコンには、ほとんどインタープリタ型BASICが付いてくるので少なくとも一度はインタープリタ型言語を使ったことがあるのではないでしょうか。

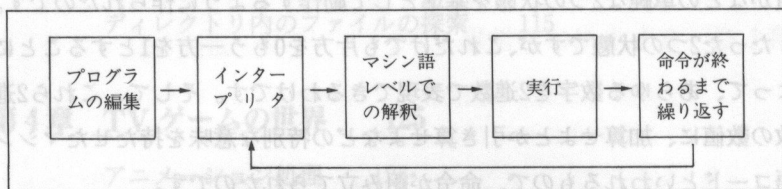



図1-1　インタープリタの実行過程

使ってみるとインタープリタ型言語は非常に手軽に使え、しかも、修正や変更もいたって簡単におこなえることがわかります。

これに対し、ソースプログラムをあらかじめ全部マシン語に翻訳して、その翻訳したマシン語のプログラムを実行するというタイプをコンパイラ型といいます。コンパイラによってマシン語に翻訳する作業をコンパイルと呼んでいますが、ソースプログラムは、コンパイラによって未定義ラベル等を含む中間的なオブジェクトファイル(拡張子OBJ)に編集されます。さらに、リンカーによって、最終的なマシン語プログラムが作られるのです。この最終的なプログラムを実行プログラム(拡張子EXE)といいます。

実行プログラムはインタープリタ型とは違って、実行時に翻訳する時間

プログラムの編集 → コンパイラ → 実行可能プログラムの編集 → 実行可能プログラムの実行

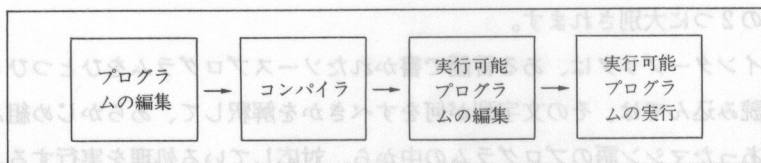



図1-2　コンパイラの実行過程

がかかりませんから、たいへん高速に実行されます。

その反面、何らかのバグを生じた場合には、翻訳される前のソースプログラムを取り出して修正し、再びコンパイルして実行を確認するという一連の面倒な作業をしなければなりません。とはいっても、得られる安全性や高速性は、何物にも代え難いといえるでしょう。

本書ではこのコンパイラ型C言語の特徴をいかしてリアルタイムゲームを作りながらC言語の可能性をさぐろうというわけです。

■　2進数（binary number）

リアルタイムゲームに限らず、高速なグラフィックス処理をするには、やはりコンピュータの世界、すなわち0と1だけの世界である2進数（binary number）の基礎的なことを知らなければなりません。2進数とはどのようなものかを知るために、ここで、4桁の2進数と10進数を対比させてみましょう。

表1-1　10進数と4桁の2進数との対応

10進数	2進数	10進数	2進数
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

いかがですか、4桁の2進数で表すことができるのは表1-1にあるように全部で0から15までの16種類です。

このように2進数は、0と1だけが並んでいるだけですから、何がどれを表しているのか人間にとっては区別がしにくいですね。

さらに2進数の世界を知るために、1桁の2進数どうしの足し算を考えてみましょう。

```
    0        1        0        1
 +  0     +  0     +  1     +  1
 -----    -----    -----    -----
    0        1        1       10
```

図1-3　1桁の2進数の足し算

はじめの3つの演算は10進数とまったく同じですが、注目したいのは最後の1+1です。これは10進数では2となりますが、2進数の世界ですから2にな

ると桁上がりを起こして10となります。十進数で2と表しても、2進数で10と表しても物の本質、すなわち、絶対的量にはなんら変わりがないというのは書くまでもありません。

ここで、先ほどの表1-1を再び参照してみましょう。2進数をもっと身近に感じるために紙とエンピツで2進数の0000から最後の1111になるまで順に1を足してみてください。

実際に計算をしてみると、なんとなくコンピュータの味け無い世界に触れたような気がしませんか。

■　2進数の単位の呼び方

さて、コンピュータ気分に浸ったところで話を先に進めましょう。我々人間の世界では物事の関係を表すのに色々な単位を用いています。たとえば重さを表すのにkg、時間を表すのには時、分、秒、などですが、コンピュータの世界でもいろいろな単位を使っています。

コンピュータの世界の基本単位である2進数1桁のことは、ビット（bit）と呼んでいます。これはbinary-digitの略です。取りうる値は0か1の2つだけしかありません。これから本書では1桁の2進数とか2桁の2進数とは言わずに、このビットの単位を使っていきます。したがって1桁の2進数は1ビット、2桁の2進数は2ビットということになります。

2進数4桁ではニブル(nibble)といいます。そして、8桁ではバイト(byte)といいますが、このバイトはコンピュータの世界ではビットと並んでよく使われる単位です。たとえば、PC-9801のメモリ上の記憶単位がバイト単位(8桁の2進数）ですし、マシン語もバイト単位で構成されています。また、英文字や、特殊文字を処理する単位もバイト単位です。

これらのほかにはワード（word）という単位がありますが、これは処理系によって4ビット、12ビット、32ビットなど、いろいろな桁数があります。PC-9801では16ビットをひとつのワードとしています。

これら4つの単位（ビット、ニブル、バイト、ワード）は、コンピュータの世界では常識として扱われていますから、その言葉の意味を覚えておかなければなりません。

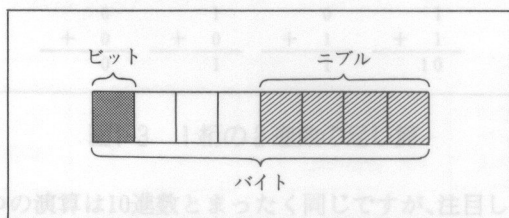



図1-4　2進数の単位

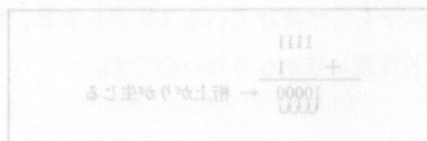
■ 文字コード

画面上に表示されている英文字や漢字はコンピュータが扱う場合に、やはり2進数として扱っています。ただ、CRT の画面上に表示された時が2進数ではなく、文字の形をしているにすぎません。

すなわちひとつひとつの文字に、2進数を1対1に対応させているのです。もちろん、2進数1桁では、0か1の2通りしか対応できませんから、ある程度十分な桁数である8桁の2進数（1バイト）を使っています。

この対応のさせ方はいろいろ考えられますが、処理系やコンピュータによってばらばらではまったく互換性がなくなりますから、そこにはちゃんと標準化がなされています。

PC-9801では英文字や特殊文字を表すコードとしてアスキーコードに準拠しています。たとえば、英文字Aのアスキーコードは2進数では01000001となります。Bは01000010、Cは01000011です。ですから、信号として01000001が送られるとCRT上に英文字のAが表示されるわけです。

なお、ソースプログラムはエディタを使って、このアスキーコードで編集します。

■　2進数と16進数

ところで、2進数を扱う時には0と1の羅列ですから人間にとってはとても苦痛です。しかも、書き表す時には、かなりのスペースを使いますので、むだが多いともいえます。

そこで2進数のおもしろい性質を使って、2進数を16進数で表記するという手品のような、まったく不思議な方法を使うわけです。ここではあっさりとその種あかしをしてみましょう。

通常よく使われるメモリの基本単位である8ビットの2進数を考えてみます。まず代表選手として、00001111という半分が1である状態を考えます。さらに、この8桁の2進数を上位4桁と下位4桁の2つに分けてみます。

```
        0 0 0 0 1 1 1 1
        ↓       ↓
  0 0 0 0         1 1 1 1
  上位4ビット     下位4ビット
```

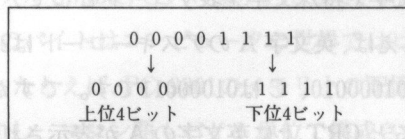



図1-5　8桁の2進数を2つに分ける

この図で、下位の2進数1111は表の1-1より、10進数では15に対応していることが分かります。

さて、この下位の4ビット1111に1を足すとどうなるでしょうか。

```
   1111
 +    1
 ------
  10000 ← 桁上がりが生じる
```

図1-6　下位4ビットに1を足す

上のように計算されますが、得られた結果は下位4ビットでは足りずに、上位4ビットの方へと桁上がりを生じることが分かります。

このことは、2進数の4桁が集まって、ひとつの数字を構成すると見立て

ることによって、16になれば上位の位へと桁上がりを起こす16進数と同じ動きをしているとみなせるのです。したがって8ビットの2進数は、2桁の16進数として表記できるというわけです。

ところで、数字は0から9までしかありませんから、16進表記をする時には10～15までを、アルファベットのA～Fの6文字に対応させています。これからは16進数といえばアルファベットのA～Fも数字の仲間に加えてあげてください。

また、16進数と10進数との明確な区別をするために、一般的にはHexadecimal（16進数）の頭文字のHを16進数の後ろに付けて表記しています。なお、C言語で扱う数値は16進数の前に0xを付記することになっています。

PC-9801ではモニタを使ってメモリの内容を表示したり、メモリに数値を書き込んだりすることができますが、この時に威力を発揮するのがこの16進表記です。また、BASICのPEEK文やPOKE文でも16進表記で2進数を扱うことができます。

このように2進数と16進数は密接な関係にあるのですが、これはあくまでも見かけ上であって、コンピュータは2進数として扱っているのです。

■ マシンの心臓部を探る

マシンの心臓部、すなわちCPU (Central Processing Unit) のことですが、やはりC言語を十分に利用するにはCPUの特徴ぐらいは知っておくべきでしょう。PC-9801シリーズでは、8086系CPUが使われています。このCPUの特徴は、なんといってもセグメントによるメモリ管理の方法です。

この概念は、メモリをCPUが一度に数えられる範囲で分割し、その分割単位で管理するという考え方です。この分割単位をセグメントと呼び、そのセグメント内の相対的な位置をオフセットといいます。

ですからメモリを直接参照する場合には、8086系CPU上でプログラムが動いている限り、どのセグメントを使うのかというセグメントアドレスと、そのセグメント内の相対的なオフセットアドレスの2つを与える作業が必要となってきます。PC-9801上でTURBO Cをインストールする時に、メモリモデルを指定したと思いますが、このメモリモデルもプログラムでセグメントをどのように使っていくのかの指定に他なりません。

C言語は高級言語であるにもかかわらず、メモリを直接アクセスすることができるようになっています。当然のことながら直接メモリをアクセスする場合には、セグメントアドレスとセグメント内のオフセットアドレスを設定することになります。

さて、本書がテーマとして掲げているTVゲームは、実は、このメモリに直接アクセスする作業にほかならないのです。

■ PC-9801のG.V.RAMとは

PC-9801のメモリには、グラフィックスを表示するための特別なメモリが用意されています。これをグラフィックビデオラム(以下G.V.RAMと略記)といいますが、このメモリは、他のメモリと同様にバイト(8ビットの2進数)単位で数値を記憶させせることができます。ただ、他のメモリと異なるのは、記憶した数値のビットイメージ(2進数の1桁)が画面上のドット(点)に対応しているということです。

すなわち、CRT画面でグラフィックス処理をするということは、このG.V.RAMに数値を書き込むということなのです。

G.V.RAMは、色別に4つのグループに分かれます。それぞれをプレーンと呼んでいますが、初期状態ではプレーン0は青で、セグメントアドレスA800$_{H}$、オフセットアドレス0$_{H}$から始まっています。一般的にセグメントアドレスとオフセットアドレスは:で区切って表記していますから、この場合には、A800:0000$_{H}$となります。また、各プレーンの大きさは8000$_{H}$バイトとなっています。

同様に、プレーン1が赤で、アドレスはB000:0000$_{H}$～B000:7FFF$_{H}$、プレー

	0	1	2	…………	77	78	79
0	0000$_{H}$	0001$_{H}$	0002$_{H}$	…………	004D$_{H}$	004E$_{H}$	004F$_{H}$
1	0050$_{H}$	0051$_{H}$	0052$_{H}$	…………	009D$_{H}$	009E$_{H}$	009F$_{H}$
⋮	⋮			TV画面（640×400ドットモード）			⋮
398	7C60$_{H}$	7C61$_{H}$	7C62$_{H}$	…………	7CAD$_{H}$	7CAE$_{H}$	7CAF$_{H}$
399	7CB0$_{H}$	7CB1$_{H}$	7CB2$_{H}$	…………	7CFD$_{H}$	7CFE$_{H}$	7CFF$_{H}$

図1-7　640×400ドットモード表示

ン2が緑で、アドレスはB800:0000$_{H}$～B800:7FFF$_{H}$、そして、プレーン3が輝度用で、アドレスはE000:0000$_{H}$～E000:7FFF$_{H}$となります。

ここで、各オフセットアドレスが0000$_{H}$～7FFF$_{H}$となるように、セグメントアドレスを設定してあることに注意してください。

この場合の、各プレーンのオフセットアドレスとTV画面との対応は次のようになっています。なお、実際に画面に表示されるメモリの範囲は、グラフィックスシステムに640×400ドットモードを選択した場合、オフセットアドレスで0000$_{H}$～7CFF$_{H}$（図1-7）となります。

	0	1	2	…………………	77	78	79
0	0000$_{H}$	0001$_{H}$	0002$_{H}$	…………………	004D$_{H}$	004E$_{H}$	004F$_{H}$
1	0050$_{H}$	0051$_{H}$	0052$_{H}$	…………………	009D$_{H}$	009E$_{H}$	009F$_{H}$
・	・						・
・	・						・
			TV画面（640×200ドットモード）				
・	・						・
・	・						・
198	3DE0$_{H}$	3DE1$_{H}$	3DE2$_{H}$	…………………	3E2D$_{H}$	3E2E$_{H}$	3E2F$_{H}$
199	3E30$_{H}$	3E31$_{H}$	3E32$_{H}$	…………………	3E7D$_{H}$	3E7E$_{H}$	3E7F$_{H}$

図1-8　640×200ドットモード第1ページ表示

	0	1	2	…………………	77	78	79
0	3E80$_{H}$	3E81$_{H}$	3E82$_{H}$	…………………	3ECD$_{H}$	3ECE$_{H}$	3ECF$_{H}$
1	3ED0$_{H}$	3ED1$_{H}$	3ED2$_{H}$	…………………	3F1D$_{H}$	3F1E$_{H}$	3F1F$_{H}$
・	・						・
・	・						・
・	・						・
			TV画面（640×200ドットモード）				
・	・						・
・	・						・
・	・						・
198	7C60$_{H}$	7C61$_{H}$	7C62$_{H}$	…………………	7CAD$_{H}$	7CAE$_{H}$	7CAF$_{H}$
199	7CB0$_{H}$	7CB1$_{H}$	7CB2$_{H}$	…………………	7CFD$_{H}$	7CFE$_{H}$	7CFF$_{H}$

図1-9　640×200ドットモード第2ページ表示

また、640×200ドットモードの画面に表示されるメモリの範囲はオフセットアドレスで0000_H〜$3E7F_H$(図1-8)、または、$3E80_H$〜$7CFF_H$(図1-9)となります。どちらの範囲を使うかは、ページ指定によって選択します。

プレーンが4つ（青、赤、緑、輝度）ということは単純に考えると、4種類の色しか出せないようにも思えますが、実は、各プレーンの表示アドレスが重なった（同じオフセットアドレスとなる）場合には、別の色を発色する仕組みになっているのです。この重なり方は全部で16種類ありますから、必然的に、色としては16種類が使えることになります。また、これら16種類の色は、固定されているわけではなく、4096色の中から任意の色を設定することができるようにもなっています。

さて、せっかくコンパイラを使っているのに長々と2進数やメモリの話ばかりでは興味も半減かもしれませんね。そろそろ、0と1だけの世界から抜け出すことにしましょう。

	0	1	2	……	77	78	79
0	0000H	0001H	0002H	……	004DH	004EH	004FH
1	[illegible]	[illegible]	[illegible]	……	[illegible]	[illegible]	[illegible]
⋮	⋮			TV画面（640×200ドットモード）			⋮
198	3DE0H	3DE1H	3DE2H	……	3E2DH	3E2EH	3E2FH
199	3E30H	3E31H	3E32H	……	3E7DH	3E7EH	3E7FH

図1-8　640×200ドットモード第1ページ表示

	0	1	2	……	77	78	79
0	3E80H	3E81H	3E82H	……	3ECDH	3ECEH	3ECFH
1	3ED0H	3ED1H	3ED2H	……	3F1DH	3F1EH	3F1FH
⋮	⋮			TV画面（640×200ドットモード）			⋮
198	7C60H	7C61H	7C62H	……	7CADH	7CAEH	7CAFH
199	7CB0H	7CB1H	7CB2H	……	7CFDH	7CFEH	7CFFH

図1-9　640×200ドットモード第2ページ表示

第2章　実践グラフィックス基礎編

グラフィックス処理の醍醐味はなんといっても、TVゲームに代表される2次元＋αの世界を創造することでしょう。

新しい世界を創造するのですから、それこそ、神様のような気分が味わえるわけです。しかも、それが製品化されるとなると、もうこたえられません。

この章では、グラフィックスの最も基本となるドット（点）に焦点を当てて話を進めていきます。大海も一滴の水からできているように、あらゆるグラフィックス処理の基本単位はドットですから、このドットを自由に扱うことができれば、グラフィックス処理も自在にこなせることになるのです。

では、TURBO Cを使うための、簡単な下準備から始めることにしましょう。

■ グラフィックス処理を始める前に

TURBO C に限らず、他の MS-DOS 上で動いているシステムにとって、グラフィックス処理を始める場合には、必ずグラフィックスシステムの初期化作業をしなければなりません。

というのも、MS-DOS では、グラフィックスシステムもひとつのデバイスとみなしているため、BASIC のように初めからグラフィックスシステムが使える状態ではないのです。

一般的には、MS-DOS 付属のグラフィックスドライバをデバイスドライバに組み込んで制御することになります。

これに対し、TURBO C では、MS-DOS のグラフィックスドライバとは無関係に、独自の BGI（Boaland Graphics Interface）というライブラリを用意しています。これらのドライバには、次の3種類があります。

PC98.BGI
PC98GRCG.BGI
PC98EGC.BGI

これは、PC-9801シリーズに、3種類のグラフィックスシステムがあるからです。PC-9801シリーズのグラフィックスシステムは、PC-9801U/UV/VF/VM から、グラフィックチャージャー（GRCG）が、VX からはエンハンストグラフィックチャージャー（EGC）が登載されました。

ですから、ユーザーのシステムに EGC が登載されていれば PC98EGC.BGI を、GRCG が登載されていれば PC98GRCG.BGI を、そして、GRCG も EGC もなければ PC98.BGI というグラフィックスドライバを使うことになります。

なお、グラフィックスシステムが何かを判定する関数 detectgraph() が用意されていますから、特にユーザーがどのシステムを使っているかを意

識することなしに、これらのドライバを使うこともできます。

ところで、これらの＊.BGIというファイルは、そのままの形ではなく、マスターディスクのDISK 3にあるBGI.ARC内に圧縮して格納されていますから、使用する場合には圧縮前のファイルに戻さなければなりません。圧縮前のファイルに戻すといっても、難しいことではなく、同じディスクにあるUNPACK.EXEを使って簡単に復元することができます。

次にAドライブからBドライブへのファイルの展開例を示しておきますから参考にしてください。

まず、DISK 3をドライブAに、ワークディスクをドライブBにセットし、次のように実行します。

```
BGI.ARCの展開例
A>UNPACK BGI.ARC B:*.BGI ↵
```

展開した＊.BGIのファイルは、以下のプログラムではカレントディレクトリにあるものとして扱っていきます。

グラフィックス関係の関数を使うためには、この他に、TURBO C付属のライブラリファイルGRAPHICS.LIBと、ヘッダファイルGRAPHICS.Hが必要となりますから、所定のディレクトリに用意しておいてください。なお、本書に掲載されているプログラムだけをプログラミングするならば、メモリモデルはスモールモデルとなります。

■ グラフィックス処理を開始するには

さて、これでグラフィックス処理の準備がやっと整いました。さっそくTURBO Cの関数を使って、グラフィックスシステムの初期化作業をする関数を作ってみましょう。

LIST2-0.C

```
/*
        グラフィックスシステムの初期化
*/
#include <graphics.h>          /* 必要とされるヘッダファイルをincludeする */
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int gdriver,                              /* グラフィックスモード格納用 */
    gmode;                                /* グラフィックスドライバ格納用 */

void grphinit(void)
{
    int ercode;                           /* 使用変数の宣言 */
    detectgraph(&gdriver, &gmode);        /* ドライバ&モードの設定 */
    initgraph(&gdriver, &gmode, "");      /* グラフィックスシステムの初期化 */
    ercode = graphresult();               /* エラーコードの取得 */
    if(ercode != grOk) {                  /* エラーであればメッセージを出す */
        printf("¥nグラフィックスエラー： %s¥n", grapherrormsg(ercode));
        printf("Press any key¥n");
        getch();                          /* キーの入力待ち */
        exit(1);                          /* MS-DOSへ */
    }
}
```

このプログラムの、実行文の始まりにある関数detectgraph()で、最適なグラフィックスドライバとモードを決定しています。ドライバとモードを格納する変数は他の関数でも参照することが予想されますから、関数の外で定義します。

次のinitgraph()という関数が、ここでメインとなるグラフィックスシス

テムを初期化している関数です。関数initgraph()の3番目の引数""は、先に出てきたグラフィックスドライバの＊.BGIの格納されているディレクトリパスを示しているのですが、この例のように、なにも指定しない場合にはカレントディレクトリに、グラフィックスドライバが入っていることを意味しています。

initgraph()の実行結果は、関数graphresult()で得られますから、その結果にしたがって、エラー処理をします。

ここでは、単純にメッセージを表示して関数exit(1)で、呼び出された親プロセスへ制御を戻しています。もし、エラーメッセージでプログラムが終了した場合には、もう一度、各ファイルの存在を確かめてください。

■ far指定のポインタ変数

では、実際にG.V.RAMへアクセスしてみましょう。G.V.RAMは特殊なメモリであることは、1章でも述べましたが、ここでもう一度G.V.RAMの整理をしてみます。

表2-1 G.V.RAMアドレスとプレーンとの対応

プレーン	開始セグメントアドレス	開始オフセットアドレス	VRAMの大きさ	対応色
第0プレーン	$A800_H$	0000_H	8000_H	青色
第1プレーン	$B000_H$	0000_H	8000_H	赤色
第2プレーン	$B800_H$	0000_H	8000_H	緑色
第3プレーン	$E000_H$	0000_H	8000_H	輝度

表2-1に示したように、G.V.RAMにはプレーンが4つ（初期のPC-9801では青色、赤色、緑色の3つ）あります。これらの4つのプレーンにより、ひとつの画面が構成されるのです。また、PC-9801シリーズでは、この画面が第一画面と第二画面の二組（PC9801、PC-9801Uでは第一画面のみ）用意されており、ポート$A4_H$，$A6_H$でコントロールしています。

表2-2 画面のI/Oコントロール

出力値＼ポート値	$A4_H$	$A6_H$
0	第一画面表示	第一画面アクセス
1	第二画面表示	第二画面アクセス

それぞれのプレーンの大きさは8000_Hバイトなのですが、画面上に表示される範囲は、640×400モードでは、0～$7CFF_H$となり、640×200モードではその半分となります。また、第0～第2プレーンの3プレーンだけを使うと、8色モードとなり、4プレーン全部使うと16色モードとなります。

各メモリは8ビット単位で構成され、そのビットイメージがTV画面上

のドット（点）に対応しています。

以上が、1章でのグラフィックビデオラム（G.V.RAM）に関しての概要です。なお、これら4つのプレーンは、以後、初期状態の色別に呼ぶことにします。ですから、プレーン0は、Blue（青）の頭文字をとってB面、プレーン1は、Red（赤）のRをとってR面、プレーン2はGreen（緑）のG面、そして、プレーン3はIntensity（輝度）のI面と呼ぶことにします。このほうが、0〜3と呼ぶよりも感覚的に色の構成を把握しやすいからです。

さて、C言語でこれらG.V.RAMへアクセスするためには各プレーンにたいするポインタを介してアクセスすることになります。

ポインタとは字の通りに何かを指している変数です。変数を指していたり関数を指していたり色々ですが、ここではG.V.RAMというメモリを指していることになります。しかも、セグメントアドレスとオフセットアドレスを設定しなければなりませんから、ポインタは必然的にfarタイプのポインタとなります。

次のLIST2-1が、各G.V.RAMに対するfarタイプのポインタ変数を設定するための関数です。

LIST2-1.C

```
/*
 char型とint型のG.V.RAMへのfarポインタの作成
*/

#include <dos.h>                    /* 必要とされるヘッダファイルをincludeする */

char far *c_blue;                   /* char型B面用ポインタの宣言 */
int  far *i_blue;                   /* int 型B面用ポインタの宣言 */
char far *c_red;                    /* char型R面用ポインタの宣言 */
int  far *i_red;                    /* int 型R面用ポインタの宣言 */
char far *c_green;                  /* char型G面用ポインタの宣言 */
int  far *i_green;                  /* int 型G面用ポインタの宣言 */
char far *c_itsty;                  /* char型I面用ポインタの宣言 */
int  far *i_itsty;                  /* int 型I面用ポインタの宣言 */

void pointer_set(void)
{
```

```
    i_blue  = MK_FP(0xa800, 0);      /* char型B面用ポインタの作成 */
    c_blue  = MK_FP(0xa800, 0);      /* int 型B面用ポインタの作成 */
    i_red   = MK_FP(0xb000, 0);      /* char型R面用ポインタの作成 */
    c_red   = MK_FP(0xb000, 0);      /* int 型R面用ポインタの作成 */
    i_green = MK_FP(0xb800, 0);      /* char型G面用ポインタの作成 */
    c_green = MK_FP(0xb800, 0);      /* int 型G面用ポインタの作成 */
    i_itsty = MK_FP(0xe000, 0);      /* char型I面用ポインタの作成 */
    c_itsty = MK_FP(0xe000, 0);      /* int 型I面用ポインタの作成 */
}
```

ここで、farタイプのポインタ変数を定義した所が、関数の外部であることに注意してください。定義したポインタは、色々な関数で使いますから、変数の可視性を考慮して、関数の外部で定義します。すなわち、他の関数で参照されることを前提としているわけです。

farタイプのポインタ変数に対するアドレスは、TURBO Cの関数MK_FP()を使って設定します。また、ひとつのプレーンに対してint型とchar型の二種類の変数を用意しています。

このように、ひとつのプレーンに対してポインタを二種類にしたのは、メモリが1バイト（8ビット）単位で数値を記憶することに起因します。すなわち、メモリに対する最小のアクセス単位は8ビット長のchar型となるため、char型は最も細かいグラフィックス処理をするのに適しているわけです。そして、int型はこの場合16ビット長ですから、メモリの記憶単位を一度に2つ処理するためにグラフィックス処理の高速化が図れるというメリットがあるのです。

これらのG.V.RAMへのポインタは、いわば、油絵における絵筆のようなものです。絵筆は色別にすれば、青、赤、緑、輝度、の4種類となります。それぞれに、対応したポインタを、blue，red，green，そしてintensityは長すぎますからitstyと名付け、int型のiとchar型のcを頭に付けてあります。

■ 色の混ぜ合わせ

さて、これらの4つの色は混ぜることもできます。といっても、絵の具のようにパレット上で混ぜ合わせるわけではありません。混ぜようとする各ポインタの同じオフセットアドレスに、データを書き込むことにより実現します。

たとえば、赤と青のプレーンの同じオフセットアドレスに書き込めば、紫色が発色するのです。これら色の混ぜ合わせ方は、表2-3のように16通りあります。

この表2-3のように、4つのプレーンの重なり方は16通りありますが、ここでPC-9801シリーズでは4096色が使えることを思いだしてください。実は、この16種類の色は仮りの色だったのです。ですから、これら16種類のプレーンの重なり方に対して、この表の通りに色を割り付けてあっただけのことなのです。しかし、まったく色がばらばらでは、話の進めようがありませんから、本書ではこの初期状態の色を基準に話を進めていきます。

表2-3

青	赤	緑	輝度	色	青	赤	緑	輝度	色
×	×	×	×	黒	×	×	×	○	薄い黒
○	×	×	×	青	○	×	×	○	薄い青
×	○	×	×	赤	×	○	×	○	薄い赤
○	○	×	×	紫	○	○	×	○	薄い紫
×	×	○	×	緑	×	×	○	○	明い緑
○	×	○	×	水色	○	×	○	○	明い水色
×	○	○	×	黄色	×	○	○	○	明い黄色
○	○	○	×	グレー	○	○	○	○	明い白

(注) 8色モードの時は、青、赤、緑の3プレーンが使われます。

さて、これら16通りの色、すなわち絵筆の色を把握したら、次は筆の動かし方です。では次の意味を考えてみてください。

```
c_blue + 1;   /* 8ビット長のchar型のfarポインタ変数を1増やす */
i_blue + 1;   /* 16ビット長のchar型のfarポインタ変数を1増やす */
```

ここで、i_blue, c_blueは共に、LIST2-1で設定したB面に対するfar指定のポインタ変数です。B面は初期状態では青に対応しています。両者共に、B面を指しているポインタです。

それぞれは、まったく同じように見えますが、これらの違いは何でしょうか？ それはポインタの指し示しているフィールドの扱い方です。すなわち、プレーン0のG.V.RAMを8ビット長のchar型で取り扱うのか16ビット長のint型で扱うのかという違いなのです。この違いだけはしっかりと把握しないと混乱をまねきますので注意してください。

ここに示した式は、式としてはどちらも変数を単純に＋1するという意味ですが、この場合の変数はポインタ変数ですから式の意味するところが変わってきます。ポインタ変数の場合、1増えることは、フィールド内の次のデータを指すことになるのです。

ですから、フィールドを8ビット長のchar型とすれば、次の8ビットのデータを指し、16ビット長のint型であれば、次の16ビット長のデータを指していることになります。もし、関数のエントリーを指しているポインタであれば、次の関数を指すことになるのです。

プログラムでは見かけ上同じに見えても、結果的に、メモリの刻み方は16ビット長のほうが、8ビット長の二倍の刻みとなります。

■ ドットから直線へ

言葉だけではわかりにくいので、実際にプログラムを組みながら、これらのこと（筆の動かし方）を考えてみましょう。

まず、グラフィックス処理の記念すべき初めの一歩として、画面の一番上に、横方向の直線を描くプログラムを作ってみました。メイン関数とプロジェクトファイルは次のようになります。

LIST2-2.C

```
/*
  char型のポインタを使って横に青い直線を描く
*/

#include <graphics.h>           /* 必要とされるヘッダファイルをincludeする */
#include <conio.h>

extern void grfinit(void);              /* 外部関数の宣言 */
extern void pointer_set(void);
extern char far *c_blue;                /* char型のB面用ポインタの宣言 */

void main(void)
{
    int i;                              /* 使用する変数の宣言 */
    grphinit();                         /* グラフィックスシステムを初期化する */
    pointer_set();                      /* ポインタをセットする */
    for(i = 0; i < 80; ++i) {           /* 1ライン分の0xffを描き込む */
        *(c_blue + i) = 0xff;           /* B面へ0xffを描き込む */
    }
    getch();                            /* キーの入力待ち */
    closegraph();                       /* グラフィックスシステムの終了とする */
}
```

LIST2-2.PRJ

```
LIST2-0
LIST2-1
LIST2-2
```

ここでは、プロジェクトファイル名を、LIST2-2.PRJ としましたが、これは適当に設定してください。プロジェクトファイルを TURBO C へ登録(GRPH+P) したら、コンパイル&実行です。以後、プロジェクトファイル名は、main 関数のファイル名に、"PRJ"という拡張子を付けたものとします。なお、プロジェクトファイルはメイン関数とは別のファイルとしてディスク上にあらかじめ保存しておきます。

実行してみるとCRT 画面の上方に青い横線が描けたはずです。さて、画面上の点をドットいいますが、このドットはG.V.RAM のメモリのビットイメージに対応しているというのは、もう知っていますね。ドット(点)の集まりは1本の線になるのは書くまでもありませんが、ここでメモリと画面との対応をもう一度、図1-7で確認してください。

図1-7から、メモリの並びは画面上横方向に対応し、画面のひとつのラインはメモリ総数0x50バイト(十進数で80)で構成されることがわかります。また、0x50バイトを超えた時には、メモリは連続していますが、画面上は不連続となり次のラインの左側へ移っています。このことは、画面上Y方向にメモリを刻んだ場合、0x50バイトのメモリ増加になることを示しています。

以上のことを踏まえ、LIST2-2を参照してみると、どのようなプロセスで青い線を描いたのかがよくわかると思います。

ところで、LIST2-2ではポインタ変数を直接動かす(変数の数値を増やす)のではなく、i という変数を作用させて間接的に動かしています。こうすれば、各ポインタ変数の値は常に一定値となり、TV 画面左上に固定されることになるからです。いわば、版画における見当がつくということなのです。

さて、実際にG.V.RAM メモリに描き込んでいる命令は*(c_blue + i) = 0xff です。LIST2-1で定義した青色の絵筆に相当するポインタ変数は、i_blue または、c_blue でした。どちらを使っても青い線を描くことはできます。ここで問題となるのが、これらポインタの扱い方です。LIST2-2では c_blue を使いましたから、例として i_blue を使ったもの

を次に示します。なお、16進数1桁は1章でもやりましたが4ビットに相当しています。すなわち、16進数1桁で画面上の4ドット（点4つ分）に対応しているわけです。

LIST2-3.C

```
/*
    int型のポインタを使って横に青い直線を描く
*/

#include <graphics.h>           /* 必要とされるヘッダファイルをincludeする */
#include <conio.h>

extern void grfinit(void);              /* 外部関数の宣言 */
extern void pointer_set(void);
extern int far *i_blue;                 /* int型のB面用ポインタの宣言 */

void main(void)
{
    int i,j;                            /* 使用する変数の宣言 */
    grphinit();                         /* グラフィックスシステムを初期化する */
    pointer_set();                      /* ポインタをセットする */
    for(i = 0; i < 40; ++i) {           /* 1ライン分0xffffを描き込む */
        *(i_blue + i) = 0xffff;         /* B面へ0xffffを描き込む */
    }
    getch();                            /* キーの入力待ち */
    closegraph();                       /* グラフィックスシステムの終了とする */
}
```

LIST2-3.PRJ

```
LIST2-0
LIST2-1
LIST2-3
```

この例では、ポインタに16ビット長のint型を使い筆を動かすループ回数はLIST2-2の半分の40とし、さらに、ポインタの指しているフィールドに格納する数値は、16ビット長の数値で、2進数で表すとすべてが1となる0xffff（十進数で65535）としています。

このように、ポインタ変数を換えた分、ループ回数が半分となりました

から、約半分のスピードで処理したことになります。LIST2-2、LIST2-3とを比較すると、これらポインタ変数のc_blue、i_blueという絵筆の動かし方の違いがよく分かるのではないでしょうか。

■ 色の混ぜ合わせ実践編

次に、色の混ぜ合わせを、プログラムを通して実際に試してみます。LIST2-4です。

LIST2-4.C

```
/*
   色の混ぜ合わせ
*/

#include <graphics.h>              /* 必要とされるヘッダファイルをincludeする */
#include <conio.h>

extern void grfinit(void);                 /* 外部関数の宣言 */
extern void pointer_set(void);

extern int far *i_blue;                    /* int型B面用ポインタの宣言 */
extern int far *i_red;                     /* int型R面用ポインタの宣言 */
extern int far *i_green;                   /* int型G面用ポインタの宣言 */

void main(void)
{
   int i;                                  /* 使用する変数の宣言 */
   grphinit();                             /* グラフィックスシステムを初期化する */
   pointer_set();                          /* 各ポインタを初期化する */
   for(i = 0; i < 40; ++i) {               /* 1ライン分0xffffを描き込む */
        *(i_blue  + i) = 0xffff;           /* B面へ0xffffを描き込む */
        *(i_red   + i) = 0xffff;           /* R面へ0xffffを描き込む */
   }
   getch();                                /* キーの入力待ち */
   closegraph();                           /* グラフィックスシステムの終了とする */
}
```

LIST2-4.PRJ

```
LIST2-0
LIST2-1
LIST2-4
```

宣言部を除けば、プログラムとしては、LIST2-3にR面をアクセスするための一行を加えただけです。

このプログラムが実行することは、最初に青い線を描き、青い線と画面上同じ位置（同じオフセットアドレス）に、赤い線を描くという作業です。

実際に、このプログラムを実行してみると画面上に表れた直線は紫色となっているはずです。これは、画面上同じ位置に、青、赤、を発色したため、青と赤を混ぜ合わせたことになって紫色の直線となったわけです。ここでのポイントは各色別の絵筆が、CRT画面上、同じ位置に線を描いたということです。すなわち各ポインタに加算している数値が同じなのです。

このように色を混ぜ合わせるプロセスは、CRT画面上同じ位置の必要なプレーンに対して、数値を書き込むことによって簡単に実現できるのです。なお、この各プレーンの組み合せ方による色合いは表2-3を参照してください。

参考のために、LIST2-4.Cに輝度用のポインタを使った場合や、その他、色々な組み合せで調べみるのもよい方法だと思います。

■ 縦方向に直線を描くには

横方向に直線を描きましたから、次に縦方向の直線を描くことを考えてみましょう。また、この時のCRT画面上の直線の位置としては、一番左側とします。

まず、方針を立ててみようではありませんか。図1-7を参照すると、画面上一番左の一番上に対応するオフセットアドレスは0であることがわかります。

ですから、G.V.RAM上のオフセットアドレスの初期値は0から始めればよさそうです。では、その次の連続したドットはどこのオフセットアドレスに書き込めばよいのでしょうか。やはり図1-7より、メモリ上0x50刻んだ所が縦方向に進んだ次に位置するメモリですから、オフセットアドレスは0x50であることがわかります。同様に縦方向に線を描く場合のアドレスは、

0，0x50，0xA0，0xF0，……

となります。したがって、アドレス変化はG.V.RAMの初項0に単純に0x50を加算していくプログラムとなりそうです。

これでメモリの刻み方の方針がたったようなので。次に、このG.V.RAMへどのような数値を格納するかが問題となります。

ところで、メモリアクセスの最小単位は8ビット長のchar型です。すなわちひとつのメモリで8ドット分まとめて記憶するのです。横方向では、これでもまったく問題がなかったのですが縦方向の場合簡単ではなさそうです。

実は、グラフィックス処理をする時にはメモリのアクセス単位が8ビット長であるということがメリットでもあり、また、最大の問題点ともなってくるのです。ここでLIST2-5.Cを参照してください。

LIST2-5.Cでは、ポインタにchar型を使っています。最後までの方針

LIST2-5.C

```
/*
        縦に青い直線を描く
*/

#include <graphics.h>                  /* 必要とされるヘッダファイルをincludeする */
#include <conio.h>

extern void grfinit(void);             /* 外部関数の宣言 */
extern void pointer_set(void);
extern char far *c_blue;               /* char型のB面用ポインタの宣言 */


void main(void)
{
    int i, j;                          /* 使用する変数の宣言 */
    grphinit();                        /* グラフィックスシステムを初期化する */
    pointer_set();                     /* ポインタをセットする */
                                       /* 縦方向へ0x80を描き込む */
    for(i = 0, j = 0; i < 399; ++i, j += 80) {
        *(c_blue + j) |= 0x80;         /* B面へ0x80を描き込む */
    }
    getch();                           /* キーの入力待ち */
    closegraph();                      /* グラフィックスシステムの終了とする */
}
```

LIST2-5.PRJ

```
LIST2-0
LIST2-1
LIST2-5
```

をたてる前にプログラムを示しましたが、ジックリとプログラムを読んでみてください。

ここで、ひとつのG.V.RAMメモリに格納されている8ビットの画面上の対応を調べてみましょう。

図2-1で示したようにひとつのメモリ上8ビットのデータの内、最上位のビットは画面上では左側に位置しています。

LIST2-5.Cでは一番左側の線を描くことを目的としていますから、CRT上一番左側に位置するメモリに格納されている数値の最上位ビット

だけが1であればよいわけです。2進表記すると次のようになります。

1	0	0	0	0	0	0	0

これを16進表記したものは0x80となります。したがって、この数値をメモリに書き込めばよいのです。

しかし、さらに注意が必要なのです。メモリに0x80をそのまま書き込んだ場合、目的のビット以外は0ですから、1ドットだけを書き込みたいのに、他の無関係な7ドット分も0、すなわち、同じメモリにある他のビット情報を壊してしまうことになるのです。

ここでひとつの工夫が必要になってきます。これには論理演算という巧妙なテクニックを使うのです。もし、LIST2-5.C のプログラムを読んでこのテクニックに気が付いた方は、もう本書を卒業してもよいくらいのテクニックなのです。

サンプルプログラムの中でG.V.RAMへ数値を書き込んでいる部分に着目してください。ここでは、この問題点を解決するために、メモリに対してORをとっていたのです。こうすればメモリ上の目的のビットだけを1に、すなわちドットを描くことができるのです。

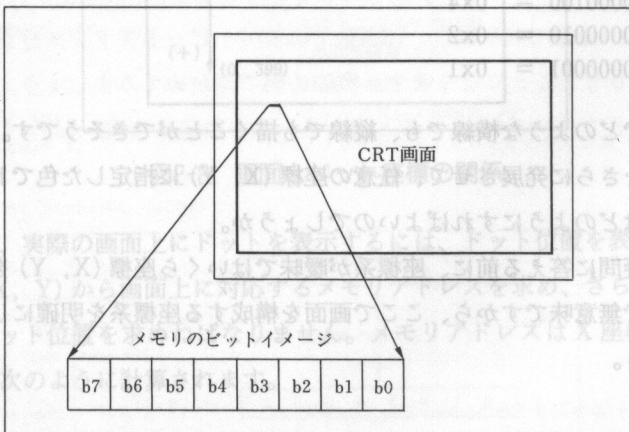



図2-1　メモリのビットイメージ

メモリとの演算にORを利用する

```
          10000000  =  0x80
OR )      00000001  =  0x01
          10000001  =  0x81
```

縦線を描く場合には、この作業を画面上、縦方向に展開することになります。G.V.RAMへのポインタのメモリ上Y方向増分は、初めの方針の通り、0x50ですから、メモリを0x50毎に刻んで、メモリに格納すべき数値である0x80とのORをとれば、目的の縦線が描けることになります。

これらを実現したのがLIST2-5.Cだったのです。では、右方向に1ドットずらした縦の直線を描く場合にはどうなるのでしょうか。これは、ビットが右にひとつずれたデータである0x40を描き込むことになります。さらに右にずらすと0x20となります。参考のために、ひとつのビットだけが1となる8桁の2進数と、16進数の対応を次に示します。

```
10000000 = 0x80
01000000 = 0x40
00100000 = 0x20
00010000 = 0x10
00001000 =  0x8
00000100 =  0x4
00000010 =  0x2
00000001 =  0x1
```

これでどのような横線でも、縦線でも描くことができそうです。では、これらをさらに発展させて、任意の座標（X, Y）に指定した色でドットを描くにはどのようにすればよいのでしょうか。

この疑問に答える前に、座標系が曖昧ではいくら座標（X, Y）を与えたところで無意味ですから、ここで画面を構成する座標系を明確にしておきましょう。

■ ドット座標の定義

今、画面横方向をX軸、鉛直方向にY軸を考えてみてください。この場合、メモリの構造から画面左上がこの座標系の原点となります。また、X軸の＋方向は画面に向かって右方向となり、Y軸の＋方向は画面下方向とするのが一般的となります。

ところで、メモリの記憶単位が1バイト（8ビット）であることは、もう知っていますね。また、横の1ラインは0x50（十進数では80）バイトで構成されていることも知っています。ですから、X軸の刻みをドット単位、すなわち、G.V.RAM上のビット単位で考えると、X軸方向のドット総数は8×80＝640となります。

また画面上のライン数の最大値は400ですからY軸方向のドット総数は400（640×200ドットモード選択時は200）となります。

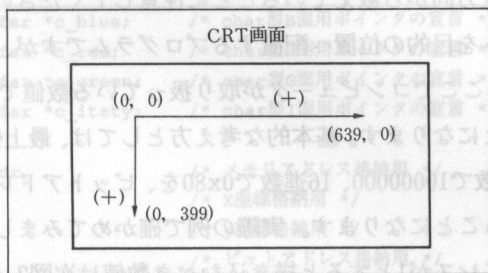



図2-2　画面とドット座標の関係

さて、実際の画面上にドットを表示するには、ドット位置を表している座標（X，Y）から画面上に対応するメモリアドレスを求め、さらにメモリ内のビット位置を求めねばなりません。メモリアドレスはX座標とY座標から次のように計算されます。

メモリアドレス＝X÷8＋Y×80　（小数点以下切捨て）

ただ、これだけではバイト単位のメモリアドレスが求まるだけです。まだ、メモリの1記憶単位である8ビット上の、どの位置にビットを配置するかというビットアドレスを求める問題が残っています。

このビットの位置は、先ほどのドットのX座標を8で割った余りとして求めることができます。次の図を参照してください。

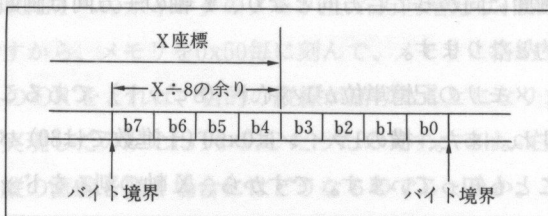



図2-3 メモリとビットアドレスの対応

ビットアドレスの取りうる値の範囲は8で割った余りですから、必然的に0〜7となります。この時、ビットアドレスはメモリの記憶単位である8ビットの数値の上位方向から数えていることに注意してください。

さて、ビットを目的の位置へ配置するプログラムですが、これを簡単にするためには、ここでコンピュータが取り扱っている数値である、2進数の性質を使うことになります。基本的な考え方としては、最上位が1となっている数値、2進数で10000000、16進数で0x80を、ビットアドレス分だけ右方向に移動させることになります。実際の例で確かめてみましょう。たとえば、ビットアドレスが1とすると描き込むべき数値は次図2-4のようになります。

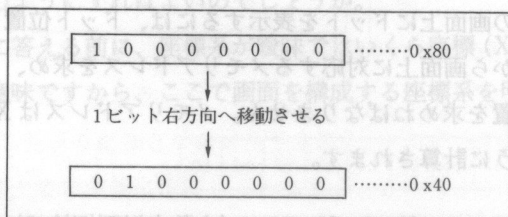



図2-4 ビットアドレスが1の場合の例

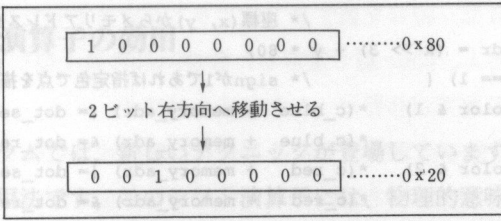



図2-5 ビットアドレスが2の場合の例

さらに、ビットアドレスが2の場合には図2-5のようになります。

ですから、画面上にドットを表示するには、X座標を8で割った余りとして求められるビットアドレス分、0x80を右方向にシフトして求められる数値と、あらかじめ求めておいたG.V.RAM上のメモリアドレスとの間で、論理演算ORを実行するということになるのです。これらをプログラムにしたのが次のLIST2-6です。

LIST2-6.C

```
extern char far *c_blue;        /* char型B面用ポインタの宣言 */
extern char far *c_red;         /* char型R面用ポインタの宣言 */
extern char far *c_green;       /* char型G面用ポインタの宣言 */
extern char far *c_itsty;       /* char型I面用ポインタの宣言 */

int memory_adr,                 /* メモリアドレス格納用 */
    x,                          /* x座標格納用 */
    y,                          /* y座標格納用 */
    bit_adr,                    /* ビットアドレス格納用 */
    sign = 1;                   /* ドットをセットするか否かのサイン用 */

void dot_plot(int color)
{
    int i, j;                   /* 使用変数の宣言 */
    unsigned char
        dot_set = 0x80,
        dot_res = 0;

    bit_adr   = x & 7;          /* 変数bit_adrへx÷8の余りを求める */
    dot_set >>= bit_adr;        /* 0x80をビットアドレス分右方向へシフトする */
    dot_res  ^= dot_set;        /* リセット用データの作成 */
```

```
                                /* 座標(x, y)からメモリアドレスを求める */
    memory_adr = (x >> 3) + y * 80;
    if(sign == 1) {                 /* signが1であれば指定色で点を描く */
        if(color & 1)     *(c_blue  + memory_adr) |= dot_set;
        else              *(c_blue  + memory_adr) &= dot_res;
        if(color & 2)     *(c_red   + memory_adr) |= dot_set;
        else              *(c_red   + memory_adr) &= dot_res;
        if(color & 4)     *(c_green + memory_adr) |= dot_set;
        else              *(c_green + memory_adr) &= dot_res;
        if(color & 8)     *(c_itsty + memory_adr) |= dot_set;
        else              *(c_itsty + memory_adr) &= dot_res;
    }
}
```

■ シフト演算子の効用

このプログラムでは、新しいテクニックが登場しています。それは、シフト演算子の用法です。このシフト演算子には、物理的意味と算術的な意味があります。

物理的意味としては、単純に数値の桁をずらすということになります。そして、この場合の数値はコンピュータにとっては2進数のことですから、2進数の数値の桁をずらすことになるのです。

もう説明するまでもありませんが、メモリに格納する数値の初期値を0x80（2進数では10000000となる）とし、X座標を8で割った剰余で右方向にシフトさせれば、正しいドット位置が設定できるというわけです。

なお、8で割った剰余は2進数の性質から、7とのANDにより求めることができます。ちょっとしたテクニックとして覚えておきましょう。

シフト演算子には、もうひとつ算術的な意味があります。ご存じのように数値の桁をシフトするということは、右方向にシフトさせればその数値の基数で割ることを意味し、左方向にシフトすれば基数倍することになるのです。

2進数の基数は2ですから、右方向に一回シフトさせれば2で割ることを意味し、左方向に一回シフトさせれば2倍になるというわけです。

この性質を使って（X÷8）を実現しているのが、LIST2-6.Cの（X >> 3）です。これは3回右方向にシフトするという意味ですから結果的に（X÷8）が求められることになるのです。なお、この演算は、単純に割り算するよりもはるかに高速になります。

ところで、このシフト演算子を使う場合には、注意しなければならない部分があります。まず、LIST2-6.Cで、初期値0x80を格納している変数dot_setの宣言部分に着目してください。

（unsinged char）となっています。これは、この変数の数値の符号は考慮しないことを宣言しているのです。実は、符号を考慮すると、右方向へシ

フトした場合には、符号を表す最上位ビットがシフトした所に入ってくるのです。もちろん、char型で宣言した数値0x80の最上位ビットは1ですから、(unsinged)宣言が必要だったというわけです。

このように、シフト演算子を使う場合は、使用する目的を十分に考慮すると同時に、変数を宣言する時には、符号の有無にも配慮しなければなりません。

■ 色の成分チェック

これでドットの位置は決定できますから、次に、色の指定について話しを進めることにします。LIST2-6.Cでは、指定した色がどのプレーンを使って表現されるのかという判断を、論理演算の&を利用して実現しています。では、このプロセスはどのように実行されているのかを考えてみましょう。

まず、色の種類は0〜15でした。これらの数値は当然のことながらCPUは2進数として扱っています。これらを2進数で表現した数値を次に示しますから確認してください。

```
 0 …… 00000000    8 …… 00001000
 1 …… 00000001    9 …… 00001001
 2 …… 00000010   10 …… 00001010
 3 …… 00000011   11 …… 00001011
 4 …… 00000100   12 …… 00001100
 5 …… 00000101   13 …… 00001101
 6 …… 00000110   14 …… 00001110
 7 …… 00000111   15 …… 00001111
```

CPUは、このように0〜15の色番号を2進数として認識するわけですが、ここでおもしろいのは、色を構成するプレーンの発色の有る無しと、色番号を表す数値のビットが1対1に対応しているということでしょう。すなわち、一番右側のビットが1であればBプレーンが発色していることになり、右から2番目のビットが1であればRプレーンが、同様に3番目はGプレーン、4番目はIプレーンが発色しているというわけです。

ですから変数colorに指定した色が、Bプレーンを含むか否かのチェックは次の論理演算の結果をチェックすればよいことになります。

color & 00000001 ＝ 1 でBプレーンの発色有り

color & 00000001 ＝ 0 でBプレーンの発色無し

同様にR，G，Iプレーンは次のようになります。

color & 00000010 ＝ 1 でRプレーンの発色有り

color & 00000010 ＝ 0 でRプレーンの発色無し

color & 00000100 ＝ 1 でGプレーンの発色有り

color & 00000100 ＝ 0 でGプレーンの発色無し

color & 00001000 ＝ 1 でIプレーンの発色有り

color & 00001000 ＝ 0 でIプレーンの発色無し

さて、ここでif文の動作を思い出してください。

```
if（条件式）{
    ≪条件が真となる時の処理≫;
}
else        {
    ≪条件が偽となる時の処理≫;
}
```

条件式の評価は、（条件式≠0）で真となり、（条件式＝0）で偽となりますから、簡単に色の成分に従った処理ができるのです。すなわち、式の演算の結果が真(≠0)であれば対応するプレーンのビットをセットし、偽(＝0）であれば対応するプレーンのビットをリセットするわけです。

■ キーの入力を知るには

LIST2-6.Cの関数は、任意の（X，Y）座標に指定した色でドット（点）を描き込む関数でした。そこで、この関数を利用して、キーの入力に従って次々と座標を変化させながらドットを打っていくことを実現させれば、どのような直線でも描けそうです。というわけで、キー入力に従って座標を更新する関数を定義してみましょう。

LIST2-7.C

```
extern int
    x,                                    /* x座標格納用 */
    y;                                    /* y座標格納用 */

xycset(
    int xmin,                             /* x方向最小値 */
    int xmax,                             /* x方向最大値 */
    int ymin,                             /* y方向最小値 */
    int ymax)                             /* y方向最大値 */
{
    int r2;                               /* 使用変数の宣言 */

    switch(r2 = getch()) {                /* キー情報に従って座標を更新する */
        case '1':
            ++y;
            --x;
            break;
        case '2':
            ++y;
            break;
        case '3':
            ++y;
            ++x;
            break;
        case '4':
            --x;
            break;
        case '6':
            ++x;
```

```
                break;
            case '7':
                --y;
                --x;
                break;
            case '8':
                --y;
                break;
            case '9':
                --y;
                ++x;
                break;
            default: break;
        }
        if(y > ymax)     y = ymax;          /* y座標の最大値のチェック */
        if(y < ymin)     y = ymin;          /* y座標の最小値のチェック */
        if(x > xmax)     x = xmax;          /* x座標の最大値のチェック */
        if(x < xmin)     x = xmin;          /* x座標の最小値のチェック */
        return(r2);                         /* キー情報を関数の戻り値とする */
}
```

ここでは、TURBO Cの関数getch()でキーボードからの情報を得ています。そして、方向別の処理をする為に、テンキーの1, 2, 3, 4, 6, 7, 8, 9を対応させて、座標（X, Y）の更新処理をしています。また、せっかく取得したキー情報は、関数の戻り値として返しています。

これで、キー情報に従って座標を更新する関数と、座標に従ってドットを表示する関数がそろいましたから、それぞれをリンクさせてコンパイルおよび実行といきましょう。main関数とプロジェクトファイルは次のようになります。

LIST2-8.C

```
#include <graphics.h>           /* 必要とされるヘッダファイルをincludeする */

extern void grphinit(void);        /* 外部関数の宣言 */
extern void pointer_set(void);
extern void dot_plot(int);
extern int xycset(int, int, int, int);
```

```
void main(void)                      /* main関数の定義 */
{
    int keydata = 0,                 /* キー情報格納用 */
        xmax    = 639,               /* x座標の最大値 */
        ymax    = 399,               /* y座標の最大値 */
        xmin    = 0,                 /* x座標の最小値 */
        ymin    = 0,                 /* y座標の最小値 */
        color   = 2;                 /* 描写する点の色の指定 */
    grphinit();                      /* グラフィックスシステムの初期化 */
    pointer_set();                   /* ポインタの設定 */
    while(keydata != 0x1b) {         /* [ESC]キーのチェック */
                                     /* 点の位置およびキーデータ設定 */
        keydata = xycset(xmin, xmax, ymin, ymax);
        dot_plot(color);             /* 点の描写 */
    }
    closegraph();                    /* グラフィックスシステムの終了 */
}
```

LIST2-8.PRJ
LIST2-0 LIST2-1 LIST2-6 LIST2-7 LIST2-8

■ カーソルの表示

LIST2-8.PRJを起動してみると、テンキーに従った直線は自由に引けるのですが、ドットを表示する現在位置が不明なのが非常に見づらいことが分かると思います。そこで次の課題として、現在のドット位置を示すカーソル表示へと話を進めることにしましょう。

まずカーソルの表示で考えなければならないのは、カーソルと、すでに表示されている絵との重ね合わせです。この時のカーソルはグラフィックスを描く為のカーソルですから、他の絵と同様にG.V.RAMへ表示しなければなりません。当然のことながら、カーソルを表示すればすでにG.V.RAM上に存在している絵は壊されてしまいます。カーソルを移動させると後には古いカーソルの残骸が残っているわけです。そこで、なんらかの工夫をして古いカーソルの部分を元の状態に復元することになります。

ここでは最も単純な背景とのXOR(排他的論理和)をとる方法を試みてみます。次にXORの真理値表を示しますから参照してください。

表2-4 XORの真理値

X	Y	X (+) Y
0	0	0
0	1	1
1	0	1
1	1	0

(+):XOR

この真理値表をよくみるとXとYが異なれば演算結果が1となり、XとYが同じであれば、0となっていることがわかります。

今、カーソルのデータをX、背景の絵をYとし、XとYとのXORをほどこした演算結果をZとして考えてみましょう。

この場合、Zがカーソル表示によって破壊されたデータということになります。ここで問題となるのは、背景YをXとの演算XORによって破壊したデータZをいかにして元のYという背景のデータに復元するかとい

うことです。すなわちXとZからYが求まればよいわけですが、実は、これはXORの性質から次のように簡単に求められるのです。

Y = X(+)Z

すなわち、カーソルのデータXと背景YとのXORをほどこした部分Zに、再び、カーソルのデータXとのXORをほどこせば、元の背景Yが復元されるというわけです。ですから、カーソルの表示、背景の復元、いずれの処理にしても単純に背景とのXORをとるプログラムのみで実現できるわけです。

LIST2-9.C

```
extern char far *c_blue;            /* B面へのポインタの宣言 */
extern char far *c_red;             /* R面へのポインタの宣言 */
extern char far *c_green;           /* G面へのポインタの宣言 */
extern char far *c_itsty;           /* I面へのポインタの宣言 */
int csr_on = 0;                     /* カーソルのオン/オフの状態を示す */

void cursorxor(
    int j,                          /* カーソルのメモリアドレス */
    int s,                          /* カーソルのビットアドレス */
    int c )                         /* カーソルの色 */
{
    static unsigned int
    crsptn[] = {                    /* カーソル形状データ */
        0x8000,                     /*  1000000000000000  */
        0xc000,                     /*  1100000000000000  */
        0xc000,                     /*  1100000000000000  */
        0xe000,                     /*  1110000000000000  */
        0xe000,                     /*  1110000000000000  */
        0xf000,                     /*  1111000000000000  */
        0xf000,                     /*  1111000000000000  */
        0xf800,                     /*  1111100000000000  */
        0xf800,                     /*  1111100000000000  */
        0xfc00,                     /*  1111110000000000  */
        0xfc00,                     /*  1111110000000000  */
        0xfe00,                     /*  1111111000000000  */
        0xff00,                     /*  1111111100000000  */
        0x1800,                     /*  0001100000000000  */
```

```
        0x1800,                         /*  0001100000000000  */
        0x1800                          /*  0001100000000000  */
    };
    union {
        unsigned int  crs;              /* カーソルデータのシフト用 */
        unsigned char crs2[2];
    } pt;
    int i;                              /* 使用変数の宣言 */

    csr_on ^= 1;                        /* カーソルの表示状態を示す */
    for(i = 0; i < 16; ++i, j+=79) {
        pt.crs = crsptn[i] >> s;
        if(c & 1) *(c_blue  + j) ^= pt.crs2[1];
        if(c & 2) *(c_red   + j) ^= pt.crs2[1];
        if(c & 4) *(c_green + j) ^= pt.crs2[1];
        if(c & 8) *(c_itsty + j) ^= pt.crs2[1];
        ++j;
        if(c & 1) *(c_blue  + j) ^= pt.crs2[0];
        if(c & 2) *(c_red   + j) ^= pt.crs2[0];
        if(c & 4) *(c_green + j) ^= pt.crs2[0];
        if(c & 8) *(c_itsty + j) ^= pt.crs2[0];
    }
}
```

カーソル形状データとしては、ドットの状態を2進数に換算して、さらに、それを16進数に変換したものを直接データとして持たせています。原始的なデザイン方法ですがカーソル程度であればこれでも十分といえるでしょう。参考までに2進数による表記をコメントとして載せてあります。

また、TURBO Cでは2進数が扱えない為に16進数としましたが、問題となるのは2進数のデータと16進数のデータとの対応でしょう。これには専用の電卓（2進演算のできるもの）などがあると便利ですが、もし無い場合には4ビットずつ区切ることによって表1-1から以外と簡単に16進数へと変換できます。

カーソル形状データはメモリの記憶単位の二倍のint型とし、シフト演算によってドット単位の移動を実現しています。この時、配列の定義としてunsigned宣言が必要となることに注意してください。

ところで、int型の数値は下位バイト、上位バイトの順でメモリへ格納されるのですが、この場合、メモリの若い方が画面左側に位置していることから処理としては工夫が必要になってきます。すなわち、このままメモリに書き込んだ場合には、上位バイトと下位バイトの左右の位置関係が逆転することになるわけです。

そこで、一旦シフトしたカーソルデータを共用体として定義した変数を介してバイト単位の数値に分割してから上位バイト、下位バイトという順番で格納しています。以上のことを踏まえて、これらを組み合わせたプロジェクトLIST2-10を実行してみましょう。

LIST2-10.C

```
#include <graphics.h>            /* 必要とされるヘッダファイルをincludeする */
#include <bios98.h>

#define  ON  1;
#define  OFF 0;

extern void grphinit(void);
extern void pointer_set(void);
extern void dot_plot(int);
extern int  xycset(int, int, int, int);
extern void cursorxor(int, int, int);

extern int
    x,                          /* x座標格納用 */
    y,                          /* y座標格納用 */
    memory_adr,                 /* メモリアドレス格納用 */
    bit_adr,                    /* ビットアドレス格納用 */
    csr_on,                     /* カーソルの現在のon/off状態を示すｻｲﾝ */
    sign;                       /* カーソル表示の管理用サイン(1でon, 0でoff) */

void main()
{
    KEY_INFO kinf;                /* キー情報格納用 */
    int keydata = 0,              /* キーデータ保存用 */
        xmax    = 639,            /* x座標最大値 */
        ymax    = 399,            /* y座標最大値 */
        xmin    = 0,              /* x座標最小値 */
        ymin    = 0,              /* y座標最小値 */
```

```
        sfsin    = 0,                /* シフトサイン用 */
        cmax     = 15,               /* カラー番号の最大値 */
        cmin     = 0,                /* カラー番号の最小値 */
        color    = 1;                /* 現在選択されているドットカラー */

    grphinit();                      /* グラフィックスシステムの初期化 */
    pointer_set();                   /* 各プレーンへのポインタの設定 */
    cursorxor(memory_adr, bit_adr, color);          /* カーソルの表示 */
    while(keydata != 0x1b) {
        keydata = xycset(xmin, xmax, ymin, ymax); /* キー情報を取得 */
        kinf.cmmd = 2;               /* bios98keyへのコマンドの設定 */
        sfsin = bios98key(&kinf);     /* シフトキーの情報を取り込む */
        if(csr_on == 1) {            /* カーソルサインがonならばカーソルの消去 */
            cursorxor(memory_adr, bit_adr, color);
        }
        switch(keydata) {        /* キー情報に従って処理をする */
            case 'x':
            case 'X':
                if(color < cmax - 1) ++color;
                break;
            case 'z':
            case 'Z':
                if(color > cmin + 1) --color;
            break;
            defaelt : break;
        }
        if(keydata != ' ') { /* [space]キーが押されていればカーソルを消去 */
            if(sfsin & 1)    sign = 0;
            else             sign = 1;
            dot_plot(color);
            cursorxor(memory_adr, bit_adr, color);
        }
    }
    closegraph();                 /* グラフィックスシステムの終了とする */
}
```

LIST2-10.PRJ

```
LIST2-0
LIST2-1
LIST2-6
LIST2-7
LIST2-9
LIST2-10
```

どうですか、カーソルが表示されたでけでもグレードがグッと上がったのではないでしょうか。実際にプロジェクト LIST2-10.PRJ を走らせて、いろいろな直線を描いてみてください。また、X，Z キーで色の変化を、スペースキーでカーソルの ON/OFF を管理するようにしてありますから、これらも同時に試してください。LIST2-10.C まで進んでくると、G.V.RAM の構成や仕組みがかなりハッキリとしてきたのではないでしょうか？

■ 割り込み処理の実際

次のチャレンジとしては、ゲームを作る上で欠かせない割り込み処理の方へと話しを進めていきましょう。

割り込みと聞くと、あまり良いイメージがありませんが、コンピュータの世界では、欠かせないものなのです。

たとえば、CPUがある処理を実行している時にストップキーが押されたとか、どこかから通信を受けたとか、割り込みは様々な要因で発生します。CPUはこれらの要因に従って、割り込みベクタテーブルをサーチし、要因別の処理へと制御を移すのです。そして、割り込み処理が終了すれば、また元の処理へと制御を移し粛々と処理を続行していきます。

これらの処理は、高速に行われるために人間にとっては割り込まれたというよりも、あたかも並列に処理が進行していくように思えます。割り込みとは、まさに、割り込みを意識させない所に意義があるのです。

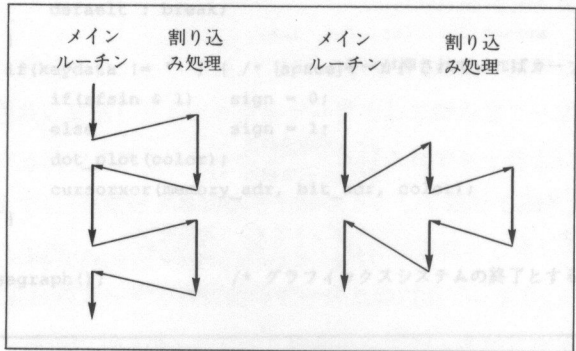



図2-6 割り込み処理の実行過程

■ マウスを組み込めば、お絵書きツールのでき上り

では、割り込み処理の入門として、ここではマウスを取り上げてみましょう。マウスの割り込み処理を利用することによって、プロジェクトLIST2-10.PRJをお絵書きツールへと変身させようというわけです。

まず、マウスを使う為には、MS-DOSのデバイスドライバにマウスドライバを組み込まなければなりません。手順としては次のようになります。

手順	内容
手順1	CONFIG.SYSへMOUSE.SYSを登録
手順2	MS-DOSシステムを再起動
手順3	マウスシステムの初期化
手順4	マウス環境の設定

図2-7 マウスドライバの組み込み手順

本書では、NEC版マウスドライバに対応させていますから、他のドライバを使う場合には各々のドライバのマニュアルを参照してください。

マウスからの信号は割り込みとして処理されますが、マウスドライバを使っている限り、特に割り込みに関する知識は要求されません。また、マウスドライバを使うといっても、直接使うわけではなく、TURBO Cに付属の関数を介して間接的に使うことになります。では、次のリストを打ち込んでください。

LIST2-11.C

```
#include <bios98.h>                /* 必要とされるヘッダファイルをincludeする */
#include <dos.h>

extern char far *c_blue;                    /* char型B面へのポインタの宣言 */
extern char far *c_red;                     /* char型R面へのポインタの宣言 */
extern char far *c_green;                   /* char型G面へのポインタの宣言 */
extern char far *c_itsty;                   /* char型I面へのポインタの宣言 */
```

```
extern void dot_plot(int);                  /* ドット描写用外部関数の宣言 */
extern void cursorxor(int, int, int); /* カーソル表示用外部関数の宣言 */

extern int                                  /* 外部変数の宣言 */
    x,                                      /* x座標格納用 */
    y,                                      /* y座標格納用 */
    color,                                  /* 指定色格納用 */
    sign,                                   /* カーソルの表示または消去を示す */
    bit_adr,                                /* ビットアドレス */
    csr_on,                                 /* カーソルの表示状態を示す */
    memory_adr;                             /* メモリアドレス */

MOUSE_INIT msint;                           /* マウス初期化用構造体定義 */
union REGS reg;                             /* レジスタ用構造体定義 */
MOUSE_INFO msinf;                           /* マウス情報格納用構造体定義 */

void msdotplot(int, int, int, unsigned, unsigned);

void mouseinit(
    unsigned int mxmin,                     /* マウスの可動範囲x方向の最小値 */
    unsigned int mxmax,                     /* マウスの可動範囲x方向の最大値 */
    unsigned int mymin,                     /* マウスの可動範囲y方向の最小値 */
    unsigned int mymax )                    /* マウスの可動範囲y方向の最大値 */
{
    msint.cmmd = 0;                         /* コマンドを初期化とする */
    bios98mouse_init(&msint);               /* マウス環境の初期化 */
    msint.cmmd = 12;                        /* コマンドを関数の登録とする */
    msint.call_cond = 0x1f;                 /* msdotplotを呼び出す条件 */
    msint.mouse_fun = msdotplot;            /* 関数名msdotplotを登録 */
    bios98mouse_init(&msint);         /* msdotplotをユーザー定義関数とする */
    msint.cmmd = 16;                  /* コマンドをx方向可動範囲の設定とする */
    msint.min_h_pos = mxmin;          /* x方向の可動範囲の最小値を指定する */
    msint.max_h_pos = mxmax;          /* x方向の可動範囲の最大値を指定する */
    bios98mouse_init(&msint);         /* 可動範囲の設定 */
    msinf.cmmd = 4;              /* コマンドをマウスカーソル位置の設定とする */
    msinf.abs_h_crs = mxmin;          /* マウスカーソルのx方向初期値を設定 */
    msinf.abs_v_crs = mymin;          /* マウスカーソルのy方向初期値を設定 */
    bios98mouse(&msinf);              /* マウスカーソルの初期位置の設定 */
    x = mxmin;                        /* ドット描写用x座標の初期化 */
    y = mymin;                        /* ドット描写用y座標の初期化 */
    bit_adr = mxmin & 7;              /* ビットアドレス初期化 */
    memory_adr = (mxmin >> 3) + mymin * 80; /* メモリアドレス初期化 */
}
```

```
void mousexyset(int x0, int y0) /* マウスカーソルの位置を設定する */
{
    msinf.cmmd = 4;               /* コマンドをマウスカーソル位置の設定とする */
    msinf.abs_h_crs = x0;         /* x方向位置をx0とする */
    msinf.abs_v_crs = y0;         /* y方向位置をy0とする */
    bios98mouse(&msinf);          /* マウスカーソル位置を座標(x0, y0)とする */
}

int msonoff = 1;                  /* マウスのオン/オフ管理用 */

void mouseterm(void)              /* マウスシステムの終了とする */
{
    msint.cmmd = 0;               /* コマンドを初期化とする */
    bios98mouse_init(&msint);     /* マウスシステムの初期化 */
}

void msdotplot(                   /* マウス用ユーザー定義関数 */
    int status,                   /* マウスの状態 */
    int lft_buton,                /* マウス左ボタンの状態 */
    int rgt_buton,                /* マウス右ボタンの状態 */
    unsigned abs_h_crs,           /* マウス用カーソルの水平軸座標 */
    unsigned abs_v_crs )          /* マウス用カーソルの垂直軸座標 */
{
    if(msonoff == 1) {            /* msonoff=1ならカーソル表示し座標を更新 */
        if(csr_on == 1) {         /* カーソルが表示状態であればカーソルを消去 */
            cursorxor(memory_adr,bit_adr,color);
        }
        x = abs_h_crs;            /* x軸カーソル位置を新たにx座標とする */
        y = abs_v_crs;            /* y軸カーソル位置を新たにy座標とする */
        if(rgt_buton == -1) {     /* 右ボタンがオンであればカラーを変更する */
            if(status & 8) {
                ++color;
                if(color == 16) color = 0;
            }
        }
        if(lft_buton == -1) sign = 1; /* 左ボタンがonなら点の描写とする */
        else                sign = 0;
        dot_plot(color);          /* signに従って点を描写する */
        cursorxor(memory_adr, bit_adr, color); /* カーソルを表示する */
    }
    else if(csr_on == 1) {        /* カーソルが表示されていればカーソルの消去 */
        cursorxor(memory_adr, bit_adr, color);
    }
}
```

LIST2-11.Cではマウスに関する基本的な関数を用意しました。各関数の機能は次のようになっています。

mouseinit …【機能】 マウスシステムの初期化及び環境の設定
【引数】 NO.1…X座標のとりうる最小値
NO.2…X座標のとりうる最大値
NO.3…Y座標のとりうる最小値
NO.4…Y座標のとりうる最大値

mousexyset …【機能】 マウスカーソルの表示位置の設定
【引数】 NO.1…カーソルのX座標
NO.2…カーソルのY座標

mouseterm …【機能】 マウスシステムの終了

msdotplot …【機能】 マウスの割り込みによって起動される関数で、カーソルの表示と、右のボタンが押されていれば、カーソル位置にドットをプロットする。また、左のボタンが押されていれば、カラーを変化させる。
【引数】 NO.1…割り込みの要因
NO.2 左ボタンの状態 (0:解放, −1:押下)
NO.3 右ボタンの状態 (0:解放, −1:押下)
NO.4…X座標
NO.5…Y座標

ここで中心となる関数はmsdotplot()ですが、これは、直接コールするのではなく、マウス割り込みによって起動される関数として登録します。すなわち、この関数が割り込み処理というわけです。また、カーソルはこの関数msdotplot()で表示するようにしてありますので、マウスドライバ独自のカーソルはOFFとしてあります。カーソルの表示及びドット表示の

プロセスは今までのものと変わりがありません。

さて、プロジェクトLIST2-10にLIST2-11を関連させたものが、プロジェクトLIST2-12.PRJです。

LIST2-12.C

```
#include <graphics.h>          /* 必要とされるヘッダファイルをincludeする */
#include <dos.h>
#include <bios98.h>

#define ON  1

extern void grphinit(void);      /* 外部関数の宣言 */
extern void pointer_set(void);
extern void dot_plot(int);
extern int xycset(int, int, int);
extern void cursorxor(int, int, int);
extern void mouseinit(
    unsigned int,
    unsigned int,
    unsigned int,
    unsigned int
);
extern void mouseterm(void);
extern void mousexyset(int, int);

extern int                  /* 外部変数の宣言 */
    x,                      /* x座標格納用 */
    y,                      /* y座標格納用 */
    memory_adr,             /* メモリアドレス */
    bit_adr,                /* ビットアドレス */
    csr_on,                 /* カーソルの表示状態を示す */
    sign;                   /* カーソル表示の管理用サイン(1でon, 0でoff) */

int color = 4;

void main()
{
    KEY_INFO kinf;              /* キー情報格納用構造体定義 */
    int sfsin,                  /* シフトキー情報 */
        keydata = 0,            /* キーデータ管理 */
        xmax = 639,             /* x座標最大値 */
```

```
        ymax = 399,                    /* y座標最大値 */
        xmin = 0,                      /* x座標最大値 */
        ymin = 0;                      /* y座標最小値 */
    grphinit();                        /* グラフィックスシステムの初期化 */
    pointer_set();                     /* 各プレーンへのポインタの設定 */
    mouseinit(xmin, xmax, ymin, ymax);         /* マウスシステムの初期化 */
    cursorxor(memory_adr, bit_adr, color); /* カーソルの表示 */
    kinf.cmmd = 2;                             /* 機能コードセット */
    while(keydata != 0x1b) {                   /* [ESC]キーチェック */
        keydata = xycset(xmin, xmax, ymin, ymax); /* キー情報セット */
        sfsin = bios98key(&kinf); /* シフトキー情報の取り込み */
        disable();                     /* 割り込み禁止 */
        if(csr_on == ON) {             /* カーソルサインがonならばカーソル消去 */
            cursorxor(memory_adr, bit_adr, color); /* カーソルの消去 */
        }
        mousexyset(x, y);              /* マウスカーソル位置の指定 */
        if(keydata != ' ') {           /* [space]キーが押されているかのチェック */
            sign = sfsin & 1;  /* シフトキー情報をセット */
            sign ^= 1;    /* シフトキー情報に従ってドットのon/offを決定 */
            dot_plot(color);   /* signに従ってドットをプロットする */
            cursorxor(memory_adr, bit_adr, color); /* カーソルの表示 */
        }
        enable();                      /* 割り込みの許可 */
    }
    mouseterm();                       /* マウスシステムの終了 */
    closegraph();                      /* グラフィックスシステムの終了とする */
}
```

LIST2-12.PRJ

```
LIST2-0
LIST2-1
LIST2-6
LIST2-7
LIST2-9
LIST2-11
LIST2-12
```

ここで注意するのは、テンキーによって座標が変更される場合です。というのも、マウス割り込みによって起動される関数msdotplot()でも座標を更新しているために、関数どうしが競合してしまうという問題が生じる

からです。

この競合という問題を避けるために、テンキーによる一連のプロセスに入る前で割り込みを禁止（disable）し、処理の終了後、割り込みを許可（enable）しています。すなわちG.V.RAMに対する処理の一本化を図るわけです。このへんの所が割り込み処理の面倒な所とも言えるでしょう。

実際に実行してみると、プロジェクトLIST2-12.PRJは、お絵書きツールの基本形になっていることがわかると思います。すなわち、これに色々な飾りを付けると、市販のツールとなるわけです。興味のある方は、ぜひチャレンジしてみてください。

これで第2章は幕となります。次の第3章では、第2章の応用として、TVゲームには必須のツールである。パターンエディタというプログラム作りにチャレンジすることにしましょう。

```
    ymax = 399;                  /* y座標最大値 */
    ymin = 0;                    /* y座標最小値 */

  pointer_set();                 /* 矢印データのポインタの設定 */
  mouseinit(xmin, xmax, ymin, ymax);      /* マウスシステムの初期化 */

  kinf.comnd = 2;                /* 機能コードセット */
  while(keydata != 0x1b) {       /* [ESC]キーのチェック */
    keydata = [illegible]
    sfain = bios98key(&kinf);  /* シフトキー情報の取り込み */
    disable();                   /* 割り込み禁止 */
    if(csr_on == ON) {         /* カーソルが[illegible] */
      cursorxor(memory_adr, bit_adr, color); /* カーソルの消去 */
    [illegible]
    [illegible]
    if(keydata != ' ') {       /* [space]キーが押されているかのチェック */
      sign = sfain & 1;  /* [illegible] */
      sign ^= 1;     /* シフトキー情報に従ってドットのon/offを決定 */
      dot_plot(color);    /* signに従ってドットをプロットする */
      cursorxor(memory_adr, bit_adr, color); /* カーソルの表示 */
    }
    enable();                    /* 割り込みの許可 */
  }
  mouseterm();                   /* マウスシステムの終了 */
  closegraph();                  /* グラフィックスシステムの終了とする */
}
```

LIST2-12.PRJ

```
LIST2-0
LIST2-1
LIST2-6
LIST2-7
LIST2-9
LIST2-11
LIST2-12
```

ここで注意するのは、テンキーによって座標が変更される場合です。というのも、マウス割り込みによって起動される関数msdotplot()でも座標を更新しているために、関数どうしが競合してしまうという問題が生じる

第3章　パターンエディタに挑戦

本章で取り上げるパターンエディタは、TVゲームにはなくてはならない必須のツールと言えるものです。しかも、パターンエディタにはTVゲームで使われる基本的なテクニックが山ほどありますから、ウォーミングアップのつもりで取り組んでください。

もちろん、本書で作りあげるゲーム用のキャラクタをデザインする為にも使いますし、その他にも色々と利用価値のあるツールとなります。では、汎用BOXルーチンへと話を進めることにしましょう。

■ BOXルーチンの作成

点の集まりは直線になり、直線の集まりは面となるわけですが、TVゲームに出てくるキャラクタは基本的にBOX型をしています。複雑に見えるパターンであっても、BOX状に並んでいるデータをくり抜くことによって表現しているのです。しかも、X方向はしばしばメモリの記憶単位である8ドットを基本構成とし、Y方向も、X方向に合わせて8ドット毎にデザインすることが多いようです。すなわち、縦横8ドットが、パターンをデザインする時の基本単位となるわけです。

ここでは、任意のG.V.RAMアドレスに、任意の色、任意の大きさで、BOX状に描写する関数を作ってみましょう。では、LIST3-0.Cです。

LIST3-0.C

```
#include <stdlib.h>

extern char far *c_blue;                /* char型B面へのポインタの宣言 */
extern char far *c_red;                 /* char型R面へのポインタの宣言 */
extern char far *c_green;               /* char型G面へのポインタの宣言 */
extern char far *c_itsty;               /* char型I面へのポインタの宣言 */

void vramdotset(
    int  memory_adr,                    /* 表示アドレス */
    int  color,                         /* 表示カラー */
    char dot_set,                       /* 表示データ */
    int  planeb,                        /* Bプレーンチェック用 */
    int  planer,                        /* Rプレーンチェック用 */
    int  planeg,                        /* Gプレーンチェック用 */
    int  planei,                        /* Iプレーンチェック用 */
    int  loopx,                         /* x方向ループ回数 */
    int  loopy,                         /* y方向ループ回数 */
    int  shift,                         /* 表示カラーシフト用 */
    int  dshift )                       /* 表示データシフト用 */
{
    char dot_res;                       /* データリセット用 */
    int  i, j;
```

```
  for(i = 0; i < loopy; ++i) {       /* Y方向のループ */
      dot_res = ~dot_set;            /* リセット用データを求める */
      for(j = 0; j < loopx; ++j) { /* X方向のループ */
          if(color&planeb) *(c_blue  + memory_adr + j) |= dot_set;
          else             *(c_blue  + memory_adr + j) &= dot_res;
          if(color&planer) *(c_red   + memory_adr + j) |= dot_set;
          else             *(c_red   + memory_adr + j) &= dot_res;
          if(color&planeg) *(c_green + memory_adr + j) |= dot_set;
          else             *(c_green + memory_adr + j) &= dot_res;
          if(color&planei) *(c_itsty + memory_adr + j) |= dot_set;
          else             *(c_itsty + memory_adr + j) &= dot_res;
      }
      memory_adr += 80;              /* 次ラインの表示アドレスを求める */
      dot_set = _rotl(dot_set, dshift);   /* 表示データをシフトする */
      color <<= shift;               /* 構成色のチェック用データを求める */
  }
}
```

LIST3-0.Cのプログラムの流れは、ドットをX方向に展開して線を、さらに、その線をY方向に展開して面としています。ですから、2章で組んだドット描写用の関数を、少し改造するだけで完成です。

特にこの関数で工夫した所といえば、グラフィックスの色を構成しているプレーンを判断するためのデータ（planeb〜i）を、引数として可変にしたということです。

このように、プレーンを選別するためのデータが可変になるだけでも、この関数の自由度はかなりアップします。また、G.V.RAMへ書き込むための変数dot_setと、色を管理している変数colorにも、dshift、shiftという引数を作用させてシフトすることが可能（=0ではシフトされない）となっている所にも注意してください。では、LIST3-0.Cを実際に利用して、画面左上に赤い色で四角形を描いてみましょう。

LIST3-1.C

```
/*
    赤い正方形を描く
*/

#include <graphics.h>
#include <dos.h>
#include <bios98.h>

extern void grphinit(void);
extern void pointer_set(void);
extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );

void main()
{
    grphinit();
    pointer_set();
    vramdotset(0, 2, 0xff, 1, 2, 4, 8, 10, 10*8, 0, 0);
    getch();
    closegraph();
}
```

LIST3-1.PRJ

```
list2-0
list2-1
list3-0
list3-1
```

さて、パターンエディタではドット単位で編集するわけですから、ドットの状態を的確に把握しなければなりません。しかし、ドットが細かいために、画面から直接ドットの状態を把握することはほとんど不可能です。そこで、このドットの状態を1ドット毎に拡大表示したいという欲求が生じるわけです。

ここで、先ほどのLIST3-0.Cでグラフィックスの色を構成するプレーンを判断する為のデータ（planeb〜i）を、なぜ、引数としなければならないのかという理由が分かったのではないかと思いますが、次の関数では、

G.V.RAM上のデータを読み込んで、色を構成するプレーンを判断する為の情報としています。これをビット単位でチェックすることによって、簡単にG.V.RAM上に展開したグラフィックスを拡大表示できるわけです。

では、このあたりの所に注意して次のLIST3-2.Cへと進んでください。

LIST3-2.C

```
#include <graphics.h>
#include <conio.h>

extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int);

extern char far *c_blue;
extern char far *c_red;
extern char far *c_green;
extern char far *c_itsty;

extern int  xmax;                          /* 編集エリアのX方向最大値 */
extern int  ymax;                          /* 編集エリアのY方向最大値 */

int wwindow = 1,                           /* 編集エリアのG.V.RAMアドレス */
    kwindow = 1 + 40 * 80 * 2 + 80 * 48;
                                        /* 拡大表示エリアのG.V.RAMアドレス */

/* 編集エリアを拡大表示する */
void zoombitdt(void)
{
    int  bitb, bitr, bitg, bitl, bitt, i, j, k, m, n, o, p;
    char bdtb, bdtr, bdtg, bdtl;

    k = kwindow + 80;
    m = wwindow;
    p = xmax / 8;
    for(o = 0; o < ymax; ++o) {
        for(n = 0; n < p; ++n) {
            bitt = 0x80;
            bitb = *(c_blue  + m);
            bitr = *(c_red   + m);
            bitg = *(c_green + m);
            bitl = *(c_itsty + m);
```

```
                ++m;
                for(j = 0, i = 0; j < 8; ++j) {
                    if(i & 1)    vramdotset( k++, bitt, 0x0e, bitb, bitr,
                                                bitg, bitl, 1, 3, 0, 0 );
                    else         vramdotset( k , bitt, 0xe0, bitb, bitr,
                                                bitg, bitl, 1, 3, 0, 0 );
                    bitt >>= 1;
                    ++i;
                }
            }
            m += 80 - p;
            k += 80 * 4 - xmax / 2;
        }
}

/* 拡大エリアの枠の描写 */
void dispwaku (void)
{
    int x0, y0;

    x0 = 5;
    y0 = 16 * 8 - 2;
    line(x0, y0, xmax * 4 + 9, y0);
    line(xmax * 4 + 9, y0, xmax * 4 + 9, y0 + ymax * 4 + 3);
    line(xmax * 4 + 9, y0 + ymax * 4 + 3, x0, y0 + ymax * 4 + 3);
    line(x0, y0 + ymax * 4 + 3, x0, y0);
}

/* 編集範囲の設定 */
void initset(void)
{
    while(1) {
        clrscr();
        printf("¥ninput box size (8 < 64) ? ");
        scanf("%d", &xmax);
        xmax=((xmax +7) / 8) * 8;
        clrscr();
        if(xmax <= 64  &&  xmax >= 8) break;
        printf("¥n***** input error *****¥n");
    }
    ymax = xmax;
}
```

このLIST3-2.Cでは、拡大表示する関数と、編集の範囲を8〜64ドットで設定する関数、そして、拡大エリアを見やすいように線で囲む関数を用意しました。

さっそく、これらの関数を組み合わせたプログラムを起動してみましょう。メイン関数とプロジェクトファイルは次のようになります。

LIST3-3.C

```
/*
    拡大表示のテスト
*/

#include <graphics.h>
#include <dos.h>
#include <bios98.h>
extern void grphinit(void);
extern void pointer_set(void);
extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );
extern void zoombitdt(void);
extern void dispwaku(void);
extern void initset(void);

extern int wwindow, kwindow;

int xmax, ymax;

void main()
{
    grphinit();
    pointer_set();
    initset();
    vramdotset(wwindow, 2, 0xff, 1, 2, 4, 8, xmax / 8, ymax, 0, 0);
    dispwaku();
    zoombitdt();
    getch();
    closegraph();
}
```

LIST3-3.PRJ
list2-0 list2-1 list3-0 list3-2 list3-3

いかがでしたか、少しはパターンエディタらしくなりました。これで編集したドットの状態を的確に把握することが可能となったわけです。

■ 割り込み処理の実際（点滅カーソルの表示）

さて、カーソル表示は前にもやりましたが、パターンエディタ用のカーソルとしては、もっとカーソルらしく等間隔に点滅するカーソルを作ってみたいものです。一口に等間隔といっても、処理としては以外と複雑になってしまいます。たとえば、プログラムで適当なウェイトをかけて、表示、点滅を繰り返す方法等が考えられますが、これでは、プログラムの動きとしてはぎこちないものとなってしまうでしょう。

そこで、等間隔でタイミングをとる割り込みを利用することになります。まず、プログラミングを行う前に、割り込みにはどの様なものがあるかを次の表で参照してみてください。

表3-1 PC-9801シリーズ割り込みベクタ

0	除算エラー
1	シングルステップ割り込み
2	NMI メモリパリティエラー
3	ブレークポイント割り込み
4	オーバーフロー割り込み
5	COPY キー割り込み
6	STOP キー割り込み
7	インターバルタイマ割り込み
8	タイマ割り込み
9	キーボード割り込み
10	V-SYNC 割り込み
⋮	⋮

以上ですが、CPU は、これら割り込みの要因に従って、メモリの0000:0000_H～0000:$03FF_H$に書き込まれている割り込みベクタを参照し、所定の処理を実行するわけです。

ですから、これらの割り込みのうち等間隔に発生する割り込みを利用することになります。ここでは10番目の V-SYNC 割り込みを利用して等間

隔に点滅するカーソルを作ってみます。V-SYNC割り込みとは、1/60秒毎のCRTの垂直帰線区間にかかる割り込みのことです。

ところで、マウスの割り込み処理の時には、TURBO Cの関数によって自動的に煩わしい一連の処理を済ませてしまいましたから、とても楽でしたが、実際には次のような手順となります。

手順1 …interrupt修飾子を用いて割り込み用関数を定義する
手順2 …対応する割り込みテーブルの元々のベクタと割り込みマスクレジスタの状態を保存する
手順3 …対応する割り込みテーブルへ、目的の関数へのベクタを登録する
手順4 …割り込みマスクレジスタの割り込みに対応するビットをクリアして割り込みを有効にする
手順5 …処理が終了する時には、割り込みベクタテーブルと割り込みマスクレジスタを元の状態に戻す

では、これらの手順に従って割り込み関数を定義してみましょう。

LIST3-4.C

```
#include <dos.h>

extern int wwindow;
extern int kwindow;
extern char far *c_blue;
extern char far *c_red;
extern char far *c_green;
extern char far *c_itsty;

extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );
void cursor(int, int, int, int);
void xorvramdt(int, int, int);

int ex      = 0,
    ey      = 0,
```

```
    kaku_adr = 0,
    color    = 2,
    dot_adr  = 0,
    half     = 0,
    crson    = 1,
    crssign  = 0,
    crsinf   = 1,
    space;
unsigned char dot_set = 0x80;
void dotcrs(void);
int counter  = 0,
    counter2 = 0;

/* 点滅カーソル割り込み関数の定義 */
void interrupt idotcsr(void)
{
    char i;

    ++counter2;
    if(crsinf == 1) {
        if(++counter == 30) counter = 0;
        if(counter > 15) crson = 0;
        else             crson = 1;
        dotcrs();
    }
    else crson = 0;
    i = 0x20;
    outportb(0, i);
    outportb(0x64, i);
}

char keeport;
void interrupt (*keepvector)();
void interrupt idotcsr(void);

/* 点滅カーソルの初期設定 */
void initcsr(void)
{
    dot_set = 0x80 >> (ex & 7);
    dot_adr = wwindow + ey * 80 + (ex >> 3);
    kaku_adr = kwindow + (ex >> 1) + ey * 80 * 4;
```

```
    half = ex & 1;
    keeport = inportb(2);
    keepvector  = getvect(10);
    setvect(10, idotcsr);
    disable();
    outportb(2, keeport & 0xfb);
    enable();
    outportb(0x64, 1);
}

/* 点滅カーソルの終了 */
void termcsr(void)
{
    setvect(10, keepvector);
    disable();
    outportb(2, keeport);
    enable();
}

/* カーソルの描写及び消去 */
void dotcrs(void)
{
    if(crson == 0 && crssign == 1) cursor( kaku_adr, half,
                                         dot_adr, dot_set );
    else if(crson == 1 && crssign == 0) {
        dot_set = 0x80 >> (ex&7);
        dot_adr = wwindow + ey * 80 + (ex >> 3);
        kaku_adr = kwindow + (ex >> 1) + ey * 80 * 4;
        half = ex & 1;
        cursor(kaku_adr, half, dot_adr, dot_set);
    }
}

/* カーソルの描写 */
void cursor(int k, int s, int dot_adr, int dot_set)
{
    crssign ^= 1;
    xorvramdt(dot_adr, dot_set, 1);
    if(s == 0) xorvramdt(k, 0xe0, 5);
    else       xorvramdt(k, 0x0e, 5);
}
```

```
/* G.V.RAMへ指定データを書き込む */
void xorvramdt(int i, int j, int k)
{
    int l;
    for(l = 0; l < k; ++l, i+=80) {
        *(c_blue  + i) ^= j;
        *(c_red   + i) ^= j;
        *(c_green + i) ^= j;
        *(c_itsty + i) ^= j;
    }
}
```

■ 割り込み関数の定義

LIST3-4.C では、(idotcsr) と名付けた関数を割り込み関数として定義しています。この例のように、割り込み関数は、interrupt 修飾子を付けて、その関数の型を void 型として宣言します。なお、この時のメモリモデルに制限はありません。

割込関数の定義

```
void interrupt test( void ) {
        :
}
```

また、LIST3-4.C では、割り込み関数を定義すると共に、カーソル表示をする関数と、ここで定義した関数を割り込みベクタテーブルへ登録する為の関数(initcsr)、そして、割り込みベクタテーブルを復元するための関数 (termcsr) も定義してあります。

では、実際に LIST3-4.C で定義した関数を、先に組んだ関数と組み合わせてみましょう。メイン関数とプロジェクトファイルは次のようになります。

LIST3-5.C

```
/*
    点滅カーソルのテスト
*/
#include <graphics.h>

extern int  wwindow;

extern void grphinit(void);
extern void pointer_set(void);
extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );
extern void zoombitdt(void);
extern void dispwaku(void);
```

```
extern void initset(void);
extern void dotcsr(int);
extern void initcsr(void);
extern void termcsr(void);
extern void interrupt idotcsr(void);

int xmax, ymax;
void main()
{
    grphinit();
    pointer_set();
    initset();
    vramdotset(wwindow, 4, 0xff, 1, 2, 4, 8, xmax / 8, ymax, 0, 0);
    dispwaku();
    zoombitdt();
    initcsr();
    getch();
    termcsr();
    closegraph();
}
```

LIST3-5.PRJ

```
list2-0
list2-1
list3-0
list3-2
list3-4
list3-5
```

■ キーの入力を知るには（高速編）

キーボードからの入力を知る方法としては今までのように、TURBO Cの関数を使用するのが一般的であり、また、最も簡単な方法ですが、すこしでも高速な処理を要求される局面では、キーボード割り込みを直接利用するという方法をとります。特にリアルタイムゲーム等では、このような手法がしばしば用いられます。また、これから作ろうとしているパターンエディタのように、キーの入力が限定されている処理でも、やはり、直接利用するとツールの応答性が良くなります。

ここでは､割り込み処理の利用サンプルとしての目的も兼ねて､このキーボード割り込みを使ってみることにしましょう。

さて､キーボードからの割り込みはV-SYNC割り込みとは異なり､等間隔ではなく、キーボード上のキーが押された、または、離された時に生じます。そして、この割り込みタイミングを利用して独自の関数へと制御を移すわけです。

この時のキーの情報は､ポートの41_Hより得られます。なお､それぞれのキーとポート41_Hより得られるデータとの対応は、表3-2を参照してください。

次のプログラムでは､使用するキーの入力に対応した処理をすると共に､得られたキー情報は､外部変数のkeydatへ格納して､現在のキー情報を他の関数からも参照できるようにしてあります。

LIST3-6.Cでは、関数keyset()をキーボード割り込みとして登録する関数initkey()と、割り込みベクタを元の状態へ復元する関数termkey()を定義してあります。特に、関数initkey()を起動した後は、関数termkey()を実行しなければ、キーの入力は限定されたままですから注意してください。では、プロジェクトLIST3-7.PRJを実行してみましょう。

表3-2　キーコード表

押した時の出力データ

						0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1
D7 D6 D5 D4	D3	D2	D1	D0		0	1	2	3	4	5	6	7
	0	0	0	0	0	ESC	Q タ	F ハ	< , ネ 、	−	.	STOP	SHIFT
	0	0	0	1	1	1 ! ヌ	W テ	G キ	> . ル ゜	/	NFER	COPY	CAPS
	0	0	1	0	2	2 " フ	E イ ィ	H ク	? / メ ・	7		f.1	カナ
	0	0	1	1	3	3 # ア ァ	R ス	J マ	_ ロ	8		f.2	GRPH
	0	1	0	0	4	4 $ ウ ゥ	T カ	K ノ	SPACE	9		f.3	CTRL
	0	1	0	1	5	5 % エ ェ	Y ン	L リ	XFER	*		f.4	
	0	1	1	0	6	6 & オ ォ	U ナ	; + レ	ROLL UP	4		f.5	
	0	1	1	1	7	7 ' ヤ ャ	I ニ	: * ケ	ROLL DOWN	5		f.6	
	1	0	0	0	8	8 (ユ ュ	O ラ	] } ム 」	INS	6		f.7	
	1	0	0	1	9	9) ヨ ョ	P セ	Z ツ ッ	DEL	+		f.8	
	1	0	1	0	A	0 ヲ ワ	@ ~ ゛	X サ	↑	1		f.9	
	1	0	1	1	B	− = ホ	[{ 。 「	C ソ	←	2		f.10	
	1	1	0	0	C	^ ヘ	⏎	V ヒ	→	3			
	1	1	0	1	D	¥ ¦ ー	A チ	B コ	↓	=			
	1	1	1	0	E	BS	S ト	N ミ	HOME CLR	0			
	1	1	1	1	F	TAB	D シ	M モ	HELP	,			
D7 D6 D5 D4	D3	D2	D1	D0		8	9	A	B	C	D	E	F
						0 0 0 1	1 0 0 1	0 1 0 1	1 1 0 1	0 0 1 1	1 0 1 1	0 1 1 1	1 1 1 1

離した時の出力データ

LIST3-6.C

```
#include <dos.h>
#include <bios98.h>

extern int
    ex,
    ey,
    xmin,
    xmax,
    ymin,
    ymax,
    space,
    counter,
    crson,
    dot_adr,
    kaku_adr,
    color,
    dot_set;

extern void dotcrs(void);
void crsmov(void);

int keydat,
    shift_key = 0,
    keysign = 0;

/* キーボード割り込み関数の定義 */
void interrupt keyset(void)
{
    #define MAKE_RIGHT         0x48
    #define MAKE_LEFT          0x46
    #define MAKE_UP            0x43
    #define MAKE_DOWN          0x4b
    #define MAKE_UP_RIGHT      0x44
    #define MAKE_UP_LEFT       0x42
    #define MAKE_DOWN_RIGHT    0x4c
    #define MAKE_DOWN_LEFT     0x4a
    #define MAKE_ESC           0x00
    #define MAKE_SPACE         0x34
    #define BREAK_SPACE        0xb4
    #define MAKE_SHIFT         0x70
    #define BREAK_SHIFT        0xf0
```

```
    int  status_port = 0x43,
         comand_port = 0x43,
         data_port   = 0x41,
         eoi         = 0;
    char command     = 0x16,
         eoidata     = 0x20;

    if(inportb(status_port) & 0x38);
    else {
        outportb(comand_port, command);
        keydat = inportb(data_port);
        if(keysign == 0) {             switch(keydat) {
                case MAKE_DOWN:
                    if(++ey > ymax - 1) ey = ymin;
                    break;
                case MAKE_LEFT:
                    if(--ex < xmin)     ex = xmax - 1;
                    break;
                case MAKE_RIGHT:
                    if(++ex > xmax - 1) ex = xmin;
                    break;
                case MAKE_UP:
                    if(--ey < ymin)     ey = ymax - 1;
                    break;
                case MAKE_DOWN_RIGHT:
                    if(++ex > xmax-1)   ex = xmin;
                    if(++ey > ymax-1)   ey = ymin;
                    break;
                case MAKE_DOWN_LEFT:
                    if(--ex < xmin)     ex = xmax - 1;
                    if(++ey > ymax - 1) ey = ymin;
                    break;
                case MAKE_UP_RIGHT:
                    if(++ex > xmax - 1) ex = xmin;
                    if(--ey < ymin)     ey = ymax - 1;
                    break;
                case MAKE_UP_LEFT:Z
                    if(--ex < xmin)     ex = xmax - 1;
                    if(--ey < ymin)     ey = ymax - 1;
                    break;
                case MAKE_SPACE:
                    space = 1;
```

```
                    break;
                case BREAK_SPACE:
                    space = 0;
                    break;
                default :
                    break;
            }
            crsmov();
            if(space == 1) {
                crson = 0;
                dotcrs();
                if(ex & 1)  vramdotset( kaku_adr + 80, color, 0x0e,
                                        1, 2, 4, 8, 1, 3, 0, 0 );
                else        vramdotset( kaku_adr + 80, color, 0xe0,
                                        1, 2, 4, 8, 1, 3, 0, 0 );
                vramdotset( dot_adr, color, dot_set,
                            1, 2, 4, 8, 1,1, 0, 0 );
                crson = 1;
                dotcrs();
            }
            counter = 0;
        }
        if(keydat == MAKE_SHIFT)  shift_key = 1;
        if(keydat == BREAK_SHIFT) shift_key = 0;
    }
    outportb(eoi, eoidata);
}

/* カーソルのオン/オフ */
void crsmov(void)
{
    crson = 0;
    dotcrs();
    crson = 1;
    dotcrs();
}

void interrupt (*keepvector9)();
void interrupt keyset(void);

/* キーボード割り込みの初期設定 */
void initkey(void)
{
```

```
    keepvector9 = getvect(9);
    setvect(9, keyset);
}

/* キーボード割り込みを元にもどす */
void termkey(void)
{
    KEY_INFO keyinf;
    setvect(9, keepvector9);
    keyinf.cmmd = 3;
    bios98key(&keyinf);
}
```

LIST3-7.C

```
/*
    キーボード割り込みのテスト
*/
#include <graphics.h>

extern void grphinit(void);
extern void pointer_set(void);
extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );
extern void zoombitdt(void);
extern void dispwaku(void);
extern void initset(void);
extern void dotcsr(int);
extern void initcsr(void);
extern void termcsr(void);
extern void initkey(void);
extern void termkey(void);
extern int xycset(int, int, int, int);
extern void interrupt idotcsr(void);

extern int
    wwindow,
    keydat,
    space;
int xmax = 64,
    ymax = 64,
    xmin = 0,
    ymin = 0;
```

```
void main()
{
    grphinit();
    pointer_set();
    initset();
    vramdotset(wwindow, 4, 0xff, 1, 2, 4, 8, xmax / 8, ymax, 0, 0);
    dispwaku();
    zoombitdt();
    initcsr();
    initkey();
    keydat = 1;
    while(keydat != 0);
    termkey();
    termcsr();
    closegraph();
}
```

LIST3-7.PRJ

```
list2-0
list2-1
list3-0
list3-2
list3-4
list3-6
list3-7
```

実際にプロジェクトLIST3-7.PRJを実行してみると、パターンエディタの基本構造がほとんどでき上がっていることがわかります。しかも、キーの操作性は合格点のはずです。

LIST3-7.PRJまで進んでくると、だいぶパターンエディタらしくなってきましたが、まだドットの色が固定されたままです。そこで、ドットの色を自由に設定できるように、新たにドットの色を選択する為の機能を付加することにしましょう。

LIST3-8.C

```
#define CTABLE  1+80*12 +80*82
#define CCURSOR 1+80*20 +80*82
#define CSIGNE  1+80*28 +80*82
```

```
extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );
extern int color, keydat;
void ccsrmov(int, int);
int kpcolor = 2;

/* カラー用カーソルの右への移動 */
void rcolor(void)
{
    if(++color > 15) color = 0;
    ccsrmov(color, kpcolor);
    kpcolor = color;
}

/* カラー用カーソルの左への移動 */
void lcolor(void)
{
    if(--color < 0) color = 15;
    ccsrmov(color, kpcolor);
    kpcolor = color;
}

/* カラー用カーソルの移動 */
void ccsrmov(int cx, int kx)
{
    xorvramdt(CCURSOR + kx, 0xfe, 7);
    xorvramdt(CCURSOR + cx, 0xfe, 7);
    vramdotset(CSIGNE, cx, 0xff, 1, 2, 4, 8, 16, 7, 0, 0);
    keydat = 1;
}

/* カラー選択用テーブルの初期セット */
void initcolor(void)
{
    int i, j;
    for(i = 0; i< 16; ++i) {
        j = CTABLE + i;
        vramdotset(j, i, 0xfe, 1, 2, 4, 8, 1, 7, 0, 0);
    }
    xorvramdt(CCURSOR + color, 0xfe, 7);
    vramdotset(CSIGNE, color, 0xff, 1, 2, 4, 8, 16, 7, 0, 0);
}
```

さて、ドットの色を指定する場合、どの様な方法を使うかが問題になります。たとえば、色と番号を対応させて、番号によって選択するという方法などが考えられます。しかし、色の数が16色にもなると、色と番号との対応を記憶するだけでも大変な作業になってきます。

そこで、色を確認しながら選択できるように、ここでは色のテーブルを準備しておいて、カーソルの移動によって指定することにしました。

LIST3-8.Cでは、色のテーブル表示用に関数initcolor()、色用のカーソル表示に関数ccsrmov()、カーソルの右方向移動用に関数rcolor()、そして、カーソルの左方向移動用に関数lcolor()の4つの関数を用意してあります。

また、ここで注意したいのは、カーソル表示用の関数ccsrmov()の実行後、キーボード割り込みによって得られた変数（keydat）の情報は必要がなくなりますから、プログラムに影響を与えない数値（keydat =1）に書き換えているということです。このような細かい配慮も忘れないようにしてください。

では、実際に実行してみましょう。メインプログラムとプロジェクトファイルは次のようになります。

LIST3-9.C

```
#include <graphics.h>
#include <dos.h>
#include <bios98.h>


#define MAKE_RIGHT 0x39
#define MAKE_LEFT  0x38


extern void grphinit(void);
extern void pointer_set(void);
extern void zoombitdt(void);
extern void dispwaku(void);
extern void initset(void);
extern void initcsr(void);
extern void termcsr(void);
```

```
extern void initkey(void);
extern void termkey(void);
extern void rcolor(void);
extern void lcolor(void);
extern void initcolor(void);

extern int
    wwindow,
    color,
    keydat;

int xmax = 64,
    ymax = 64,
    xmin = 0,
    ymin = 0;

void main()
{
    grphinit();
    pointer_set();
    initset();
    dispwaku();
    zoombitdt();
    initcsr();
    initkey();
    initcolor();
    keydat = 1;
    while(keydat != 0) {
        switch(keydat) {
            case MAKE_RIGHT:
                rcolor();
                break;
            case MAKE_LEFT:
                lcolor();
                break;
            default :
                break;
        }
    }
    termkey();
    termcsr();
    closegraph();
}
```

LIST3-9.PRJ

```
list2-0
list2-1
list3-0
list3-2
list3-4
list3-6
list3-8
list3-9
```

プロジェクトLIST3-9.PRJでは、任意のキャラクタパターンを編集できるようになったわけです。いわば木の幹ができたといえるでしょう。しかし、これだけでは、使えるツールとは御世辞にもいえません。すなわち、使える状態にするには、さらに、枝や葉を付加しなければならないのです。

次のLIST3-10.Cでは編集エリアを一定のパターンで埋める関数を組んでいくことにしましょう。

LIST3-10.C

```
extern void dotcrs(void);
extern void zoombitdt(void);
extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );


extern int
    xmax,
    ymax,
    keysign,
    crson,
    crsinf,
    counter;
void dotcrsoff(void);
void dotcrson(void);

/* 編集エリアを指定パターンで埋める */
void allpaint(int addres, int color, char data, int shift)
{
    int i, j, k, l;
```

```
    dotcrsoff();
    k = 80 - xmax / 8;
    l = xmax / 8;
    for(j = 0; j < ymax; ++j) {
        for(i = 0; i < l; ++i, ++addres) {
            vramdotset(addres, color, data, 1, 2, 4, 8, 1, 1, 0, 0);
        }
        data = _rotl(data, shift);
        addres += k;
    }
    zoombitdt();
    dotcrson();
}

/* カーソルをオンとする */
void dotcrson(void)
{
    crsinf  = 1;
    crson   = 1;
    dotcrs();
    counter = 0;
    keysign = 0;
}

/* カーソルをオフとする */
void dotcrsoff(void)
{
    keysign = 1;
    crsinf  = 0;
    crson   = 0;
    dotcrs();
}
```

ここでのポイントは、カーソルの取り扱いです。実は、左上の編集エリアには、カーソルの位置を示すドットが点滅していたのです。せっかく凝って作ったのですから、まったく気付かなかったという人はいないでしょうね。

このドットの点滅は、割り込みによって処理されています。ドットは、XORによって表示、消去を繰り返しています。

ですから、このドットの点滅を一時的にストップさせなければならないのです。このドットの点滅を管理しているのがLIST3-10.Cのdotcrson()，dotcrsoff()という関数で、単純にコールするだけとしました。

編集エリアをパターンで埋める関数はallpaint()となります。このallpaint()を利用して、編集エリアを5種類のパターンで埋めるようにしたのが次のプロジェクトLIST3-11.PRJです。なお、編集エリアを埋める時のドットの色は、現在選択されているドットの色ということにしました。

LIST3-11.C

```
#include <graphics.h>

#define MAKE_RIGHT 0x39
#define MAKE_LEFT  0x38
#define MAKE_F1    0x62
#define MAKE_F2    0x63
#define MAKE_F3    0x64
#define MAKE_F4    0x65
#define MAKE_F5    0x66

extern void grphinit(void);
extern void pointer_set(void);
extern void zoombitdt(void);
extern void dispwaku(void);
extern void initset(void);
extern void dotcsr(int);
extern void initcsr(void);
extern void termcsr(void);
extern void initkey(void);
extern void termkey(void);
extern void rcolor(void);
extern void lcolor(void);
extern void initcolor(void);
extern void allpaint(int, int, char, int);

extern int
    wwindow,
    color,
    keydat,
```

```
    space;

int xmax = 64,
    ymax = 64,
    xmin = 0,
    ymin = 0;

void main()
{
    grphinit();
    pointer_set();
    initset();
    dispwaku();
    zoombitdt();
    initcsr();
    initkey();
    initcolor();
    keydat = 1;
    while(keydat != 0) {
        switch(keydat) {
            case MAKE_RIGHT:
                rcolor();
                break;
            case MAKE_LEFT:
                lcolor();
                break;
            case MAKE_F1:
                allpaint(wwindow, color, 0xff, 0);
                break;
            case MAKE_F2:
                allpaint(wwindow, color, 0xaa, 1);
                break;
            case MAKE_F3:
                allpaint(wwindow, color, 0x55, 1);
                break;
            case MAKE_F4:
                allpaint(wwindow, color, 0xaa, 0);
                break;
            case MAKE_F5:
                allpaint(wwindow, color, 0x55, 0);
                break;
            default :
```

```
            break;
        }
    }
    termkey();
    termcsr();
    closegraph();
}
```

LIST3-11.PRJ

```
list2-0
list2-1
list3-0
list3-2
list3-4
list3-6
list3-8
list3-10
list3-11
```

さて、プロジェクトLIST3-11.PRJのキー操作は、ファンクションキーのf1〜f5となっています。それぞれ、ドットカラーを変更して試してみてください。このように、編集エリアを埋める関数が付加されただけでもかなり使い勝手がよくなったはずです。ファンクション機能をさらに増やして、色々なアイデアを付加してみるのも面白いかもしれません。

ところで、実際にパターンを編集してみると、パターンはひとつだけとは限りません。複数のパターンを編集する場合がほとんどです。しかも、以前に編集したパターンを一部分だけ変更したい場合も多々あります。そこで、編集し終ったパターンはどこかに退避させて、必要な時に編集エリアへと再び呼び出せるような構造にしておきたいものです。

この退避するエリアとしては、当然のことながらメモリがあげられますが、編集した絵を把握するという観点から、空いている画面(G.V.RAM)を利用するという方法が有利といえます。

LIST3-12.C

```
extern int
    keydat,
    xmax,
    ymax,
    wwindow,
    shift_key;

extern char far *c_blue;
extern char far *c_red;
extern char far *c_green;
extern char far *c_itsty;
extern void dotcrson(void);
extern void dotcrsoff(void);
extern void zoombitdt(void);

void abchange(int, int);
void boxcursol(int);
void boxmov(void);

int box_inf,
    box_adr = 36,
    fukugen = 10,
    selectw = 19,
    mwindow = 36,
    bx      = 0,
    by      = 0;

/*
編集パターンを指定エリアへ退避、または退避
エリアから編集エリアへパターンを読み込む
*/
void ptn_copy(void)
{
    int i, j, k, w;
    #define MAKE_READ     0x13
    #define MAKE_WRITE    0x11
    #define MAKE_FUKUGEN  0x20
    #define MAKE_ZOOM     0x29
    #define MAKE_RIGHT    0x48
    #define MAKE_LEFT     0x46
    #define MAKE_UP       0x43
```

```
#define MAKE_DOWN     0x4b

dotcrsoff();
box_inf = 1;
w = 1;
boxcursol(box_adr);
while(shift_key == 1) {
    switch(keydat) {
        case MAKE_READ:
        case MAKE_ZOOM:
            boxcursol(box_adr);
            abchange(fukugen, wwindow);
            abchange(wwindow, box_adr);
            boxcursol(box_adr);
            if(keydat == MAKE_ZOOM) zoombitdt();
            w = 0;
            keydat = 1;
            break;
        case MAKE_WRITE:
            boxcursol(box_adr);
            abchange(fukugen, box_adr);
            abchange(box_adr, wwindow);
            boxcursol(box_adr);
            w = 1;
            keydat = 1;
            break;
        case MAKE_FUKUGEN:
            boxcursol(box_adr);
            if(w == 0)        abchange(wwindow, fukugen);
            else              abchange(box_adr, fukugen);
            boxcursol(box_adr);
            keydat = 1;
            break;
        case MAKE_RIGHT:
            if(++bx > 44 - xmax / 8) bx = 0;
            boxmov();
            break;
        case MAKE_LEFT:
            if(--bx < 0) bx = 44 - xmax / 8;
            boxmov();
            break;
        case MAKE_DOWN:
```

```
                if(++by > 50 - ymax / 8) by = 0;
                boxmov();
                break;
            case MAKE_UP:
                if(--by < 0) by = 50 - ymax / 8;
                boxmov();
                break;
        }
    }
    boxcursol(box_adr);
    dotcrson();
}

/* ボックス型カーソルの移動 */
void boxmov(void)
{
    boxcursol(box_adr);
    box_adr = by * 8 * 80 + bx + mwindow;
    abchange(selectw, box_adr);
    boxcursol(box_adr);
    keydat=1;
}

extern void xorvramdt(int, int, int);

/* ボックス型カーソルの表示 */
void boxcursol(int k)
{
    int i, j;
    if(box_inf == 1) {
        j = (ymax - 1) * 80;
        for(i = 0; i < xmax / 8; ++i, ++j) {
            xorvramdt(k + i, 0xff, 1);
            xorvramdt(k + j, 0xff, 1);
        }
        j = xmax / 8 + k + 79;
        k += 80;
        xorvramdt(k, 0x80, ymax - 2);
        xorvramdt(j, 0x01, ymax - 2);
    }
    else box_inf = 1;
}
```

```
/* bからaへのパターンのコピー */
void abchange(int a, int b)
{
    int i, j, k;
    for(j = 0, k = 0; j < ymax; ++j) {
        for(i = 0; i < xmax / 8; ++i, ++k) {
            *(c_blue   + a + k) = *(c_blue   + b + k);
            *(c_red    + a + k) = *(c_red    + b + k);
            *(c_green  + a + k) = *(c_green  + b + k);
            *(c_itsty  + a + k) = *(c_itsty  + b + k);
        }
        k += 80 - xmax / 8;
    }
}
```

■ G.V.RAM上のパターンのコピー

ここで、G.V.RAM上のパターンをコピーする手順を考えてみます。まず、画面上にある編集したパターンAを単純にA′の位置へとコピーするわけですから、それぞれの、基準となるアドレスはそれぞれのパターン左上のアドレスを求めればよさそうです。となりのアドレスは単純に+1すればよく、1ライン下のアドレスはG.V.RAMの構成から+80すればよいからです。ここで問題となるのは、編集エリアのアドレスは固定ですから簡単に求まりますが、移動すべきA′のアドレスをどのように決定するかです。

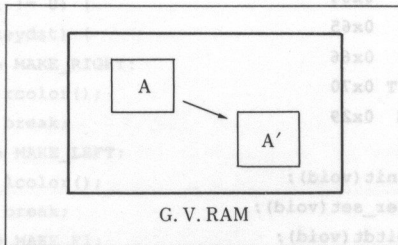



図3-1　AをA′へコピーする

表3-3　BOXカーソルの操作方法

[shift]+［2］	boxカーソルを下方向へ移動する
[shift]+［4］	boxカーソルを左方向へ移動する
[shift]+［6］	boxカーソルを右方向へ移動する
[shift]+［8］	boxカーソルを上方向へ移動する
[shift]+［W］	編集エリアのパターンをboxカーソルで示されるエリアへコピーする
[shift]+［R］	boxカーソル位置のパターンを編集エリアにコピーする
[shift]+［F］	直前のコピー動作によってコピーしたパターンを元に戻す
[shift]+［Z］	boxカーソルのパターンを編集エリアへコピーするとともに拡大表示する

そこで、LIST3-12.Cでは、感覚的にわかりやすいbox型のカーソルを移動することによってコンピュータに計算させることにしました。この時のキーの操作方法は表3-3の通りとなります。

なお、box型のカーソルを表示した時には、編集用の点滅カーソルの表示はカットしてあります。では、プロジェクトLIST3-13.PRJを起動してみましょう。

LIST3-13.C

```
#include <graphics.h>

#define MAKE_RIGHT 0x39
#define MAKE_LEFT  0x38
#define MAKE_F1    0x62
#define MAKE_F2    0x63
#define MAKE_F3    0x64
#define MAKE_F4    0x65
#define MAKE_F5    0x66
#define MAKE_SHIFT 0x70
#define MAKE_ZOOM  0x29

extern void grphinit(void);
extern void pointer_set(void);
extern void zoombitdt(void);
extern void dispwaku(void);
extern void initset(void);
extern void dotcsr(int);
extern void initcsr(void);
extern void termcsr(void);
extern void initkey(void);
extern void termkey(void);
extern void rcolor(void);
extern void lcolor(void);
extern void initcolor(void);
extern void allpaint(int, int, char, int);
extern void ptn_copy(void);

extern int
    wwindow,
    color,
    keydat,
```

```
    space;

int xmax = 64,
    ymax = 64,
    xmin = 0,
    ymin = 0;

void main()
{
    grphinit();
    pointer_set();
    initset();   dispwaku();
    zoombitdt();
    initcsr();
    initkey();
    initcolor();
    keydat = 1;
    while(keydat != 0) {
        switch(keydat) {
            case MAKE_RIGHT:
                rcolor();
                break;
            case MAKE_LEFT:
                lcolor();
                break;
            case MAKE_F1:
                allpaint(wwindow, color, 0xff, 0);
                break;
            case MAKE_F2:
                allpaint(wwindow, color, 0xaa, 1);
                break;
            case MAKE_F3:
                allpaint(wwindow, color, 0x55, 1);
                break;
            case MAKE_F4:
                allpaint(wwindow, color, 0xaa, 0);
                break;
            case MAKE_F5:
                allpaint(wwindow, color, 0x55, 0);
                break;
            case MAKE_SHIFT:
                ptn_copy();
                break;
```

```
            case MAKE_ZOOM:
                zoombitdt();
                break;
            default :
                break;
        }
    }
    termkey();
    termcsr();
    closegraph();
}
```

LIST3-13.PRJ

```
list2-0
list2-1
list3-0
list3-2
list3-4
list3-6
list3-8
list3-10
list3-12
list3-13
```

さて、これでパターンエディタの編集機能は最低限、そろったわけです。でも、せっかくここまで作ったのですから、もう少し編集機能を充実させたいものです。そこで、編集エリアの回転移動、平行移動、上下反転機能を付加することにしました。なお、左右の反転は、回転移動と上下反転によって実現することができますから、ここでは機能としては省いてあります。

LIST3-14.C

```
extern char far *c_blue;
extern char far *c_red;
extern char far *c_green;
extern char far *c_itsty;
extern int
    xmax,
```

```
    ymax,
    wwindow,
    fukugen;

extern void dotcrson(void);
extern void dotcrsoff(void);
extern void zoombitdt(void);
extern void abchange(int, int);
extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );

/* 編集パターンを回転する */
void rolling(void)
{
    int b, r, g, l, i, j, j2, k, o, o2, m, k2;
    unsigned char n, n2, nb, nr, ng, nl;

    dotcrsoff();
    abchange(fukugen, wwindow);
    k  = fukugen + xmax / 8- 1;
    o2 = wwindow;
    for(j = 0; j < ymax / 8; ++j, ++o2) {
        for(j2 = 0, n2 = 0x80; j2 < 8; ++j2, n2 >>= 1, k += 80) {
            for(i = 0, n = 0x1, o = o2, k2 = k;
                i < xmax / 8; ++i, --k2, o += 8 * 80) {
                nb = *(c_blue   + k2);
                nr = *(c_red    + k2);
                ng = *(c_green  + k2);
                nl = *(c_itsty  + k2);
                vramdotset(o, n, n2, nb, nr, ng, nl, 1, 8, 1, 0);
            }
        }
    }
    zoombitdt();
    dotcrson();
}
/* 編集パターンを右方向へ1ドット移動する */
void dotshiftr(void)
{
    int i, e, i0, iw, j, o0, ow;
    unsigned char b, r, g, l, kpb, kpr, kpg, kpl;
```

```
    dotcrsoff();
    abchange(fukugen, wwindow);
    e   = xmax / 8 - 1;
    for(j = 0, i0 = fukugen, o0 = wwindow; j < ymax;
        ++j, o0 += 80, i0 += 80) {
        kpb = *(c_blue  + i0 + e);
        kpr = *(c_red   + i0 + e);
        kpg = *(c_green + i0 + e);
        kpl = *(c_itsty + i0 + e);
        for(i = 0, iw = i0, ow = o0; i< xmax / 8; ++i, ++iw, ++ow){
            b = *(c_blue  + iw);
            r = *(c_red   + iw);
            g = *(c_green + iw);
            l = *(c_itsty + iw);
            *(c_blue  + ow) = b >> 1;
            *(c_red   + ow) = r >> 1;
            *(c_green + ow) = g >> 1;
            *(c_itsty + ow) = l >> 1;
            vramdotset( ow, 0x1, 0x80, kpb, kpr,
                        kpg, kpl, 1, 1, 0, 0 );
            kpb = b;
            kpr = r;
            kpg = g;
            kpl = l;
        }
    }
    zoombitdt();
    dotcrson();
}

/* 編集パターンを左方向へ1ドット移動する */
void dotshiftl(void)
{
    int i, e, i0, iw, j, o0, ow;
    unsigned char   b, r, g, l, kpb, kpr, kpg, kpl;

    dotcrsoff();
    e   = xmax / 8 - 1;
    for(j = 0, i0 = fukugen, o0 = wwindow; j < ymax;
        ++j, o0 += 80, i0 += 80) {
        kpb = *(c_blue  + i0);
        kpr = *(c_red   + i0);
```

```
        kpg = *(c_green + i0);
        kpl = *(c_itsty + i0);
        for(i = 0, iw = i0 + e, ow = o0 + e;i < xmax / 8;
            ++i, --iw, --ow) {
            b = *(c_blue  + iw);
            r = *(c_red   + iw);
            g = *(c_green + iw);
            l = *(c_itsty + iw);
            *(c_blue  + ow) = b << 1;
            *(c_red   + ow) = r << 1;
            *(c_green + ow) = g << 1;
            *(c_itsty + ow) = l << 1;
            vramdotset( ow, 0x80, 0x01, kpb, kpr,
                         kpg, kpl, 1, 1, 0, 0 );
            kpb = b;
            kpr = r;
            kpg = g;
            kpl = l;
        }
    }
    zoombitdt();
    dotcrson();
}

/* 編集パターンを上下に1ドット移動する */
void dotshiftud(int k)
{
    int i, j, l, m;

    dotcrsoff();
    if(k == 0) {
        j=80;
        l = wwindow + ymax * 80 - 80;
        m = fukugen;
    }
    else {
        j=-80;
        l = wwindow;
        m = fukugen + ymax * 80 - 80;
    }
    abchange(fukugen, wwindow);
    abchange(wwindow, fukugen + j);
```

```
    for(i = 0; i < xmax / 8; ++i) {
        *(c_blue  + l + i) = *(c_blue  + m + i);
        *(c_red   + l + i) = *(c_red   + m + i);
        *(c_green + l + i) = *(c_green + m + i);
        *(c_itsty + l + i) = *(c_itsty + m + i);
    }
    zoombitdt();
    dotcrson();
}

/* 編集パターンの上下を反転する */
void udhanten(void)
{
    int i, j, k, u, d;
    char m;

    dotcrsoff();
    u = 0;
    d = (ymax - 1) * 80;
    k = wwindow;
    for(j = 0; j < ymax / 2; ++j) {
        for(i= 0; i < xmax / 8; ++i, ++u, ++d) {
            m = *(c_blue + k + u);
            *(c_blue + k + u) = *(c_blue + k + d);
            *(c_blue + k + d) = m;
            m = *(c_red + k + u);
            *(c_red + k + u) = *(c_red + k + d);
            *(c_red + k + d) = m;
            m = *(c_green + k + u);
            *(c_green + k + u) = *(c_green + k + d);
            *(c_green + k + d) = m;
            m = *(c_itsty + j + u);
            *(c_itsty + k + u) = *(c_itsty + k + d);
            *(c_itsty + k + d) = m;
        }
        u += 80 - xmax / 8;
        d -= 80 + xmax / 8;
    }
    zoombitdt();
    dotcrson();
}
```

このプログラムでは、ビット操作をしていますから、コンピュータの扱っている数値が、2進数であったということを忘れてしまうと、ほとんど、プログラムとしては分かりにくいものとなってしまいます。では、さらに、次のLIST3-15.Cに進んでください。

LIST3-15.C

```
#include <graphics.h>

#define MAKE_CR     0x39
#define MAKE_CL     0x38
#define MAKE_F1     0x62
#define MAKE_F2     0x63
#define MAKE_F3     0x64
#define MAKE_F4     0x65
#define MAKE_F5     0x66
#define MAKE_SHIFT  0x70
#define MAKE_ZOOM   0x29
#define MAKE_ROLL   0x13
#define MAKE_RIGHT  0x3c
#define MAKE_LEFT   0x3b
#define MAKE_UP     0x3a
#define MAKE_DOWN   0x3d
#define MAKE_HANTEN 0x16

extern void grphinit(void);
extern void pointer_set(void);
extern void zoombitdt(void);
extern void dispwaku(void);
extern void initset(void);
extern void dotcsr(int);
extern void initcsr(void);
extern void termcsr(void);
extern void initkey(void);
extern void termkey(void);
extern void rcolor(void);
extern void lcolor(void);
extern void initcolor(void);
extern void allpaint(int, int, char, int);
extern void ptn_copy(void);
extern void rolling(void);
```

```
extern void dotshiftr(void);
extern void dotshiftl(void);
extern void dotshiftud(int);
extern void udhanten(void);

extern int
    wwindow,
    color,
    keydat,
    space;

int xmax = 64,
    ymax = 64,
    xmin = 0,
    ymin = 0;

void main()
{
    grphinit();
    pointer_set();
    initset();
    dispwaku();
    zoombitdt();
    initcsr();
    initkey();
    initcolor();
    keydat = 1;
    while(keydat != 0) {
        switch(keydat) {
            case MAKE_CR:
                rcolor();
                break;
            case MAKE_CL:
                lcolor();
                break;
            case MAKE_F1:
                allpaint(wwindow, color, 0xff, 0);
                break;
            case MAKE_F2:
                allpaint(wwindow, color, 0xaa, 1);
                break;
            case MAKE_F3:
```

```
			allpaint(wwindow, color, 0x55, 1);
			break;
		case MAKE_F4:
			allpaint(wwindow, color, 0xaa, 0);
			break;
		case MAKE_F5:
			allpaint(wwindow, color, 0x55, 0);
			break;
		case MAKE_SHIFT:
			ptn_copy();
			break;
		case MAKE_ZOOM:
			zoombitdt();
			break;
		case MAKE_ROLL:
			rolling();
			break;
		case MAKE_RIGHT:
			dotshiftr();
			break;
		case MAKE_LEFT:
			dotshiftl();
			break;
		case MAKE_UP:
			dotshiftud(0);
			break;
		case MAKE_DOWN:
			dotshiftud(1);
			break;
		case MAKE_HANTEN:
			udhanten();
			break;
		default :
			break;
		}
	}
	termkey();
	termcsr();
	closegraph();
}
```

LIST3-15.PRJ

```
list2-0
list2-1
list3-0
list3-2
list3-4
list3-6
list3-8
list3-10
list3-12
list3-14
list3-15
```

■ グラフィック画面のセーブ／ロード

パターンエディタもようやく仕上げとなります。パターンエディタとしては、プロジェクトLIST3-15.PRJで終わりですが、作成したパターンをディスクへセーブしたりロードしたりできなければ、ツールとしては不完全です。ここでは、ファイル管理について話を進めていきます。

G.V.RAM上にある編集したデータを単純にディスクへファイルとして保存したり、あるいは、ディスクにあるファイルをG.V.RAMへロードするわけですが、G.V.RAM上のデータも他のメモリ上のデータとなんら変わる所はありません。ですから、操作手順としては次のようになります。

手順1：ファイルをオープンする
手順2：ファイルをセーブ／ロードする
手順3：ファイルをクローズする

この時、ロードする場合であればロード先のメモリがG.V.RAMであり、セーブする場合であれば、データの存在するメモリがG.V.RAMであるというだけです。

なお、ファイルには管理するためのポインタであるファイルポインタがあり、すべてのファイルはこのファイルポインタを使って管理しています。LIST3-16.Cでは関数fopen()でファイルをオープンし、関数fgetc()，fputc()でディスクへの読み書きを、関数fclose()でファイルのクローズを実現しています。

LIST3-16.C

```
#include <dir.h>
#include <io.h>
#include <stdio.h>
#include <conio.h>
#include <bios98.h>
```

```
#define P_SHIFT 1

extern int mwindow;
extern char far *c_blue;
extern char far *c_red;
extern char far *c_green;
extern char far *c_itsty;

void fmes(int, char []);
void file_disp(char []);
int getname(char []);

/* グラフィック画面データをディスクへセーブする */
gsave(void) {
    char fname[64];
    FILE *sfp;
    char c;
    int i, j, k, l, fsin;

    fsin = 0;
    while(1) {
        i = 3;
        while(i) {
            printf("¥n¥ninput file name¥n");
            i = getname(fname);
            if(i == 1) {
                fsin = 2;
                break;
            }
            if(i == 2) file_disp(fname);
            else i=0;
        }
        if(fsin == 2) break;
        if(access(fname, 0) == 0) {
            printf( "¥n%s is allready exist ¥ncontinue (Y-N) ? ",
                    fname );
            while(1) {
                c = getch();
                if(c == 'n' || c == 'N') {
                    fsin = 2;
                    break;
                }
```

```
                if(c == 'Y' || c == 'y') {
                    printf("Y");
                    break;
                }
            }
            if(fsin == 2) break;
        }
        else  printf("save file name  = %s ", fname);
        sfp = fopen(fname, "wb");
        if(NULL == sfp) {
            fsin = 3;
            break;
        }
        for(i = 0, l = mwindow; i < 400; ++i, l+=80) {
            for(j = 0, k = l; j < (80 - mwindow); ++j, ++k) {
                fsin = 1;
                if(fputc(*(c_blue  + k), sfp) == EOF) break;
                if(fputc(*(c_red   + k), sfp) == EOF) break;
                if(fputc(*(c_green + k), sfp) == EOF) break;
                if(fputc(*(c_itsty + k), sfp) == EOF) break;
                fsin = 0;
            }
            if(fsin == 1) break;
        }
        fclose(sfp);
        break;
    }
    fmes(fsin, fname);
    return(fsin);
}

/* エラーメッセージの表示 */
void fmes(int fsin, char fname[])
{
    clrscr();
    gotoxy(1, 5);
    switch(fsin) {
        case 0:
            printf("* normal  end *");
            break;
        case 1:
            printf("¥7* abnormal end *");
```

```
            break;
        case 2:
            printf("* cansel *");
            break;
        case 3:
            printf("¥7 cannot open file : %s", fname);
            break;
        default:
            break;
    }
}

/* ファイル名の入力 */
int getname(char name[])
{
    int i, j;
    j = 0;
    for(i = 0; i < 40; ++i) {
        while(1) {
            name[i] = getche();
            if(name[i] == 0x8) {
                if(i > 0) --i;
                printf(" ¥b");
            }
            else break;
        }
        if(name[i] == 0x1b) {
            printf(" ");
            j = 1;
            break;
        }
        if(name[i] == 0xd) {
            printf("¥n");
            name[i] = 0;
            break;
        }
    }
    if(j != 1) {
        for(i = 0; i < 40; ++i) {
            if(name[i] == 0) break;
            if(name[i] == '*' || name[i] == '?') {
                j = 2;
```

```
                break;
            }
        }
        printf("¥ninput file name = %s¥n", name);
    }
    return(j);
}

/* ファイルの表示 */
void file_disp(char name[])
{
    int i;
    struct ffblk ffblk;

    if(findfirst(name, &ffblk, 0) == 0) {
        while(1) {
            printf("¥n%s", ffblk.ff_name);
            if(findnext(&ffblk) != 0) break;
            printf("¥t%s", ffblk.ff_name);
            if(findnext(&ffblk) !=0) break;
        }
    }
}

/* グラフィック画面データをディスクからロードする */
gload(void) {
    char fname[64];
    FILE *sfp;
    int c, i, j, k, l, s, done, fsin;
    struct ffblk ffblk;
    KEY_INFO keyinf;

    fsin = 0;
    while(1) {
        i = 3;
        while(i) {
            printf("¥n¥ninput file name¥n");
            i = getname(fname);
            if(i == 1) {
                fsin = 2;
                break;
            }
```

```
            if(i == 2) file_disp(fname);
            else i = 0;
        }
        if(fsin == 2) break;
        printf("load file name  = %s ", fname);
        sfp = fopen(fname, "rb");
        if(NULL == sfp) {
            fsin = 3;
            break;
        }
        keyinf.cmmd = 3;
        bios98key(&keyinf);
        for(i = 0, l = mwindow; i < 400; ++i, l+=80) {
            for(j = 0, k = l; j < (80 - mwindow); ++j, ++k) {
                fsin = 1;
                if((c = fgetc(sfp)) == EOF) break;
                *(c_blue + k) = c;
                if((c = fgetc(sfp)) == EOF) break;
                *(c_red + k)=c;
                if((c = fgetc(sfp)) == EOF) break;
                *(c_green + k) = c;
                if((c = fgetc(sfp)) == EOF) break;
                *(c_itsty + k) = c;
                fsin = 0;
            }
            if(fsin == 1) break;
            keyinf.cmmd = 2;
            if(bios98key(&keyinf) & P_SHIFT) break;
        }
        fclose(sfp);
        break;
    }
    keyinf.cmmd = 3;
    bios98key(&keyinf);
    fmes(fsin, fname);
    return(fsin);
}
```

■ ファイル名の入力

さて、ファイルのセーブ／ロードの手順は、それぞれTURBO Cの関数を使うだけですから問題ありませんが、LIST3-16.Cで問題となるのは、ファイル名の入力方法でしょう。人間にはミスがつきものですし、人間は、わがままですから入力の途中で中断したい場合もあるからです。

そこで、LIST3-16.Cでは比較的自由度を持たせたファイル名入力専門の関数getnameを用意ました。特徴は、一文字ずつキー情報を取り込んで引数の配列name[]へ格納していることです。というのも、[ESC]キーが入力された場合には速やかに処理を中断するようにしたいからです。また、配列ですから[BS]キーが入力された時にはインデックスを単純に−1することによって以前のキー入力値を取り消すことができるというメリットもあります。さらに、ワイルドカードの入力もチェックできるように、リターン情報をセットしています。なお、LIST3-17.Cはキーボード割り込みをコントロールするためのものです。

LIST3-17.C

```
#include <dir.h>
#include <io.h>
#include <stdio.h>
#include <conio.h>

extern termkey(void);
extern initkey(void);
extern dispwaku(void);
extern void initcolor(void);
extern void dotcrsoff(void);
extern void dotcrson(void);
extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );

extern int
   ex,
```

```
    ey;

void sideclr(int);

/* ファイル操作初期設定 */
void fstart(void)
{
    clrscr();
    dotcrsoff();
    sideclr(399);
    termkey();
    textcursor(DISP_CURSOR);
}

/* ファイル操作終了時処理 */
void fend(void)
{
    dispwaku();
    initcolor();
    dotcrson();
    textcursor(NODISP_CURSOR);
    initkey();
    outportb(0x64, 0);
}

/* 左側画面の消去 */
void sideclr(int y)
{
    vramdotset(0, 0, 0xff, 1, 2, 4, 8, 36, y, 0, 0);
}

/* 編集エリアのサイス変更用 */
void change_box(void)
{
    fstart();
    ex = 0;    ey = 0;
    initset();
    fend();
}
```

■ ディレクトリ内のファイルの探索

ファイル名にワイルドカードが入力された場合には、指定したワイルドカードに従って画面上にファイルを表示させなければなりません。というわけで、この処理を実現しているのが file _disp() という関数です。この関数の流れは、次のようになっています。

1. 関数 findfirst で、初めに一致するファイル名（ワイルドカードを含む）を探索し、存在すれば画面にそのファイル名をプリントする。
2. 関数 findnext で、次に一致するファイル名を探索する。一致するファイルが存在すればプリントし再び次に一致するファイル名を探索する。もし、一致するファイルが存在しなければ処理を終了とする。

ここでのポイントは、ファイル名の入力と、ディレクトリ内のファイル表示、ファイルポインタによるファイルの入出力管理の3つにしぼられます。この3つさえ理解できればLIST3-16.C は卒業です。では実際にこれらを組み込んだプロジェクト LIST3-18.PRJ を実行してみましょう。なお、グラフィックデータをロードする時の中断は［SHIFT］キーとしました。

LIST3-18.C

```
#include <graphics.h>
#define MAKE_CR      0x39
#define MAKE_CL      0x38
#define MAKE_F1      0x62
#define MAKE_F2      0x63
#define MAKE_F3      0x64
#define MAKE_F4      0x65
#define MAKE_F5      0x66
#define MAKE_SHIFT   0x70
#define MAKE_ZOOM    0x29
```

```
#define MAKE_ROLL    0x13
#define MAKE_RIGHT   0x3c
#define MAKE_LEFT    0x3b
#define MAKE_UP      0x3a
#define MAKE_DOWN    0x3d
#define MAKE_HANTEN 0x16
#define MAKE_LOAD    0x25
#define MAKE_SAVE    0x1e
#define MAKE_CHANGE 0x2b

extern void grphinit(void);
extern void pointer_set(void);
extern void zoombitdt(void);
extern void dispwaku(void);
extern void initset(void);
extern void dotcsr(int);
extern void initcsr(void);
extern void termcsr(void);
extern void initkey(void);
extern void termkey(void);
extern void rcolor(void);
extern void lcolor(void);
extern void initcolor(void);
extern void allpaint(int, int, char, int);
extern void ptn_copy(void);
extern void rolling(void);
extern void dotshiftr(void);
extern void dotshiftl(void);
extern void dotshiftud(int);
extern void udhanten(void);
extern void gsave(void);
extern void gload(void);
extern void fstart(void);
extern void fend(void);
extern void change_box(void);

extern int
    wwindow,
    color,
    keydat,
    space;
```

```
int xmax = 64,
    ymax = 64,
    xmin = 0,
    ymin = 0;

void main()
{
    grphinit();
    pointer_set();
    initset();
    dispwaku();
    zoombitdt();
    initcsr();
    initkey();
    initcolor();
    keydat = 1;
    while(keydat != 0) {
        switch(keydat) {
            case MAKE_CR:
                rcolor();
                break;
            case MAKE_CL:
                lcolor();
                break;
            case MAKE_F1:
                allpaint(wwindow, color, 0xff, 0);
                break;
            case MAKE_F2:
                allpaint(wwindow, color, 0xaa, 1);
                break;
            case MAKE_F3:
                allpaint(wwindow, color, 0x55, 1);
                break;
            case MAKE_F4:
                allpaint(wwindow, color, 0xaa, 0);
                break;
            case MAKE_F5:
                allpaint(wwindow, color, 0x55, 0);
                break;
            case MAKE_SHIFT:
                ptn_copy();
                break;
```

```
            case MAKE_ZOOM:
                zoombitdt();
                break;
            case MAKE_ROLL:
                rolling();
                break;
            case MAKE_RIGHT:
                dotshiftr();
                break;
            case MAKE_LEFT:
                dotshiftl();
                break;
            case MAKE_UP:
                dotshiftud(0);
                break;
            case MAKE_DOWN:
                dotshiftud(1);
                break;
            case MAKE_HANTEN:
                udhanten();
                break;
            case MAKE_LOAD:
                fstart();
                gload();
                fend();
                break;
            case MAKE_SAVE:
                fstart();
                gsave();
                fend();
                break;
            case MAKE_CHANGE:
                change_box();
                break;
            default :
                break;
        }
    }
    termkey();
    termcsr();
    closegraph();
}
```

LIST3-18.PRJ

```
list2-0
list2-1
list3-0
list3-2
list3-4
list3-6
list3-8
list3-10
list3-12
list3-14
list3-16
list3-17
list3-18
```

本書ではサンプルプログラムということで機能的には、かなり省いたものとなってしまいました。

しかし、これだけの機能があれば、本書で作成するゲームでは十分といえるでしょう。最後に、パターンエディタの締めくくりとして2章で作成した、マウスによる、お絵描きツールとドッキングさせてみましょう。当然のことながら、マウスとマウスドライバ(mouse.sys)が必要となってきます。では、メイン関数とプロジェクトファイルを打ち込んで起動してみてください。

LIST3-19.C

```
#include <graphics.h>
#include <dos.h>

#define MAKE_CR        0x39
#define MAKE_CL        0x38
#define MAKE_F1        0x62
#define MAKE_F2        0x63
#define MAKE_F3        0x64
#define MAKE_F4        0x65
#define MAKE_F5        0x66
#define MAKE_SHIFT     0x70
#define MAKE_ZOOM      0x29
#define MAKE_ROLL      0x13
#define MAKE_RIGHT     0x3c
```

```
#define MAKE_LEFT    0x3b
#define MAKE_UP      0x3a
#define MAKE_DOWN    0x3d
#define MAKE_HANTEN 0x16
#define MAKE_LOAD    0x25
#define MAKE_SAVE    0x1e
#define MAKE_CHANGE 0x2b

extern int
    wwindow,
    mwindow,
    color,
    keydat,
    memory_adr,
    msonoff,
    bit_adr,
    csr_on,
    space;

extern void grphinit(void);
extern void pointer_set(void);
extern void zoombitdt(void);
extern void dispwaku(void);
extern void initset(void);
extern void dotcsr(int);
extern void initcsr(void);
extern void termcsr(void);
extern void initkey(void);
extern void termkey(void);
extern void rcolor(void);
extern void lcolor(void);
extern void initcolor(void);
extern void allpaint(int, int, char, int);
extern void ptn_copy(void);
extern void rolling(void);
extern void dotshiftr(void);
extern void dotshiftl(void);
extern void dotshiftud(int);
extern void udhanten(void);
extern void dot_plot(int);
extern int  xycset(int, int, int, int);
extern void cursorxor(int, int, int);
```

```
extern void gsave(void);
extern void gload(void);
extern void fstart(void);
extern void fend(void);
extern void change_box(void);
extern void file_disp(void);
extern void mouseinit( unsigned int, unsigned int,
                       unsigned int, unsigned int);
extern void mouseterm(void);
extern void mousexyset(int, int);

int xmax = 64,
    ymax = 64,
    xmin = 0,
    ymin = 0;

void main()
{
    int sfsin;

    grphinit();
    pointer_set();
    initset();
    mouseinit(mwindow * 8, 639, 0, 399);
    cursorxor(memory_adr, bit_adr, color);
    dispwaku();
    zoombitdt();
    initcsr();
    initkey();
    initcolor();
    keydat = 1;
    while(keydat != 0) {
        switch(keydat) {
            case MAKE_CR:
                disable();
                if(csr_on == 1)
                    cursorxor(memory_adr, bit_adr, color);
                msonoff = 0;
                enable();
                rcolor();
                cursorxor(memory_adr, bit_adr, color);
                msonoff = 1;
```

```
            break;
        case MAKE_CL:
            disable();
            if(csr_on == 1)
                cursorxor(memory_adr, bit_adr, color);
            msonoff = 0;
            enable();
            lcolor();
            cursorxor(memory_adr, bit_adr, color);
            msonoff = 1;
            break;
        case MAKE_F1:
            allpaint(wwindow, color, 0xff, 0);
            break;
        case MAKE_F2:
            allpaint(wwindow, color, 0xaa, 1);
            break;
        case MAKE_F3:
            allpaint(wwindow, color, 0x55, 1);
            break;
        case MAKE_F4:
            allpaint(wwindow, color, 0xaa, 0);
            break;
        case MAKE_F5:
            allpaint(wwindow, color, 0x55, 0);
            break;
        case MAKE_SHIFT:
            disable();
            if(csr_on == 1)
                cursorxor(memory_adr, bit_adr, color);
            msonoff = 0;
            enable();
            ptn_copy();
            msonoff = 1;
            break;
        case MAKE_ZOOM:
            zoombitdt();
            break;
        case MAKE_ROLL:
            rolling();
            break;
        case MAKE_RIGHT:
```

```
            dotshiftr();
            break;
        case MAKE_LEFT:
            dotshiftl();
            break;
        case MAKE_UP:
            dotshiftud(0);
            break;
        case MAKE_DOWN:
            dotshiftud(1);
            break;
        case MAKE_HANTEN:
            udhanten();
            break;
        case MAKE_LOAD:
            disable();
            if(csr_on == 1)
                cursorxor(memory_adr, bit_adr, color);
            msonoff = 0;
            enable();
            fstart();
            gload();
            fend();
            msonoff = 1;
            break;
        case MAKE_SAVE:
            disable();
            if(csr_on == 1)
                cursorxor(memory_adr, bit_adr, color);
            msonoff = 0;
            enable();
            fstart();
            gsave();
            fend();
            msonoff = 1;
            break;
        case MAKE_CHANGE:
            change_box();
            break;
        default :
            break;
    }
```

```
    }
    mouseterm();
    termkey();
    termcsr();
    closegraph();
}
```

LIST3-19.PRJ

```
list2-0
list2-1
list2-6
list2-9
list2-11
list3-0
list3-2
list3-4
list3-6
list3-8
list3-10
list3-12
list3-14
list3-16
list3-17
list3-19
```

第4章　TVゲームの世界

グラフィックス処理のひとつの分野として発展してきたTVゲームですが、一口にゲームといっても、迷路型、シューティングタイプ、ロールプレイングなど様々な形態があります。中でもリアルタイムゲームの分野は、プログラムの結果を確かめながら進めることができるために、プログラミングも楽しく進めていくことができます。本章では3章のパターンエディタでデザインしたキャラクタを使って、実際にアニメーション処理を、さらに、これを発展させたリアルタイムゲームを作っていきます。

まず話を進める前に、アニメーション用の口絵パターンを用意しておきましょう。デザインしたパターンは、ひとつのファイルとしてカレントディレクトリへ格納しておいてください。プログラムを効果的に、また楽しく進めていくためにも、このような地味な作業が必要となってきます。もちろん、パターン数さえ揃っていればオリジナルのデザインでもなんら差し支えありません。なお、各キャラクタの大きさは32×32ドットとしました。

■ アニメーション処理

アニメーション処理の実際は、説明するまでもなく徐々に動いていく過程を連続して表示すれば実現できるわけです。対象となる動作をいかに細かく分解するかによって、動きのなめらかさが決まるのです。もっとも、メモリに制限のあるパソコンでは、見せ場となるパターンにより多くの動作パターンを割り付けるという手法をとります。

さて、デザインしたパターンを実際に利用するには、一枚の絵となっている画面上の絵を適当な大きさに切り取ってメモリへと格納しておかなければなりません。TURBO C には画面上の指定された範囲のビットイメージをメモリへセーブするgetimage()という関数と、セーブしたビットイメージを画面上に復元するputimage()という関数があります。これらの関数を利用するのもひとつの方法としてあげられます。

しかし、TV ゲームなどでは画面上のビットイメージを単純にセーブしてそのまま復元するよりも、これらのビットイメージをさらに加工する場合がしばしばですから、構築するシステムに合わせた独自の関数を作る必要があります。そこで、これまでの応用も兼ねて、画面上のビットイメージをメモリへセーブする関数と、画面に絵を復元する関数を作ることにしました。

メモリとは、すなわち変数のことですが、ここで画面上のデータを変数

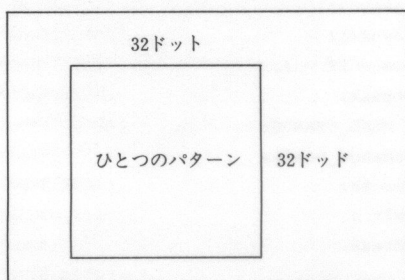



図4-1 パターンのデータ構造

へと格納することを考えてみます。口絵のデザインではひとつのパターンの大きさを32×32ドットしましたから、この場合のデータ構造は図4-1のようになります。

まずX方向です、これは汎用性を考慮すればchar型の変数配列を使いたい所ですが、スピード最優先のリアルタイムゲームということで、データをまとめて処理できる32ビット長のlong型の変数を使うことにします。ですから、X方向はlong型の変数ひとつで対応できるわけです。次にY方向ですが、これは単純にlong型の変数が32ライン重なることになります。また、プレーンは4つありますから、ひとつのパターンを構成する変数域は次のように、簡単な構造体として定義できることになります。

32×32ドットで構成するひとつのパターンを格納するための構造体定義

```
#define PTY 32
struct {
    long pb[PTY];       /* Bプレーン用データエリア */
    long pr[PTY];       /* Rプレーン用データエリア */
    long pg[PTY];       /* Gプレーン用データエリア */
    long pi[PTY];       /* Iプレーン用データエリア */
} pattern;
```

また、実際にはパターン数がひとつではありませんから、この構造体自体を配列としてパターン数分確保することにします。

32×32ドットで構成する50個のパターンを格納するための構造体定義

```
#define PTY 32
struct {
    long pb[PTY];       /* Bプレーン用データエリア */
    long pr[PTY];       /* Rプレーン用データエリア */
    long pg[PTY];       /* Gプレーン用データエリア */
    long pi[PTY];       /* Iプレーン用データエリア */
} pattern[50];          /* パターン数確保する */
```

ここでは、50個のパターンを格納できる構造体を定義しましたが、この数は適当に設定してください。

さて、このように変数の構造が確定できれば、あとは単純に画面上のデー

タを取り込むだけです。早速、ビットイメージをメモリへセーブするpattern_in()という関数と、処理としては逆のビットイメージを画面上に復元するpattern_out()という関数を作ってみましょう。

LIST4-0.C

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dir.h>

extern char far *c_blue;
extern char far *c_red;
extern char far *c_green;
extern char far *c_itsty;
#define PTY 32

/* パターン格納用構造体の定義 */
struct {
    long  dtb[PTY];
    long  dtr[PTY];
    long  dtg[PTY];
    long  dti[PTY];
} pattern[50];

/* 画面上データをメモリへセーブする */
pattern_in(unsigned int i, int j)
{
    int k;
    if(i < 50) {
        for(k = 0; k < PTY; j+=80, ++k) {
            pattern[i].dtb[k] = *(long far *)(c_blue  + j);
            pattern[i].dtr[k] = *(long far *)(c_red   + j);
            pattern[i].dtg[k] = *(long far *)(c_green + j);
            pattern[i].dti[k] = *(long far *)(c_itsty + j);
        }
        i = 0;
    }
    else    i = 1;
    return (i);
}
```

```
/* メモリから画面へパターンを復元する */
pattern_out(unsigned int i, int j)
{
    int k;
    if(i < 50) {
        for(k = 0; k < PTY; j+=80, ++k) {
            *(long far *)(c_blue  + j) = pattern[i].dtb[k];
            *(long far *)(c_red   + j) = pattern[i].dtr[k];
            *(long far *)(c_green + j) = pattern[i].dtg[k];
            *(long far *)(c_itsty + j) = pattern[i].dti[k];
        }
        i = 0;
    }
    else    i = 1;
    return (i);
}
```

これら関数の引数には、パターンNO.と32×32ドットで構成するBOX型の左上のG.V.RAMアドレスを与えることとしました。なお、パターン数の最大値である50を超えた場合には、なにも処理せずに関数の戻り値として1を返すようにしてあります。また、パターンNO.を示す引数iの定義ではunsignedとしていることに注意してください。このことにより配列を参照する時のiの範囲は0≦i<50と確定されるのです。if文の条件判断を少なくするささやかな配慮といえます。

■ キャスト演算子

LIST4-0.Cでは、配列を参照するための変数の範囲に気を配ると共に、もうひとつ重要なキャスト（cast）という概念が導入されています。

キャストとは、いったん定義した変数や式の型を、一時的に別の型へと変換することを指しています。LIST4-0.Cでは、char型のG.V.RAMへのポインタの型をパターンを格納してある変数の型であるlong型へとキャスト（型変換）しています。このキャストの特徴は、型を変換するまでは本来与えられた型が有効であるということです。当り前のようですが、このことが重要なのです。

LIST4-1.Cでは、これを利用してフィールドのアドレス計算をする時には本来のchar型で、データのやり取りはlong型というように、最も細かくG.V.RAMを操作できるchar型の特徴を生かしながら処理の高速化を図ったというわけです。

では実際に、これらの関数を利用してアニメーション処理を実行してみましょう。main()関数とプロジェクトファイルは次のようになります。

LIST4-1.C

```
/*
    アニメーション処理用メイン関数の定義
*/
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

#define NPAT 80*32

extern int pattern_in(unsigned int, int);
extern int pattern_out(unsigned int, int);
extern int gload(void);

int mwindow = 36;
```

```
void main()
{
    int i, j, k, l, m, n;
    grphinit();
    pointer_set();
    clrscr();
    printf("¥nﾊﾟﾀｰﾝ ﾃﾞｰﾀ ﾛｰﾄﾞ");
    if(gload() == 0) {
        for(k = m = n = 0, l = mwindow; k < 4; ++k, l+=NPAT) {
            for(i = 0, j = l; i < 11; ++i, j+=4) {
                n = pattern_in(m++, j);
                if(n != 0) break;
            }
            if(n != 0) break;
        }
        clrscr();
        printf("¥nｽﾀｰﾄ ﾊﾟﾀｰﾝ no.=");
        scanf("%d", &k);
        printf("ｴﾝﾄﾞ ﾊﾟﾀｰﾝ no.=");
        scanf("%d", &j);
        clrscr();
        i = k;
        do {
            pattern_out(i++, 0);
            if(i > j) i = k;
        } while(getch() != 0x1b);
    }
    else {
        printf("¥n press any key");
        getch();
    }
    closegraph();
}
```

LIST4-1.PRJ

```
list2-0
list2-1
list3-16
list4-0
list4-1
```

LIST4-1.Cを実行すると、パターンが格納してあるファイル名の入力を促すメッセージが現れます。ファイル名を指定すると、次にアニメーションのスタートNO.とエンドNO.を入力します。コマ送りは[SPACE]キーとしました。では、口絵のパターンで実験してみましょう。アニメーションのスタートNO.として7をエンドNO.として18を入力してください。[SPACE] キーを押し続けると三角形の板が回転している状態が表示されるはずです。この例ではパターンの大きさを32×32ドットとしましたから迫力に欠けますが、もっと大きなパターン用の関数を作って試してみるのもおもしろいでしょう。また、セーブしたパターンを斜め方向に復元したり、構成する色を変えたり等、セーブしたビット情報を色々と加工してみるのもよいでしょう。

■ マップエディタに挑戦

さて、ここでLIST4-0.Cのおもしろい応用を考えてみましょう。それは、CRT画面の中に2次元の世界を作り出すためのマップエディタというツールです。マップエディタというのはパターンエディタと並んでTVゲームには欠かせないツールとなります。しかも、LIST4-0.Cを使えば簡単に作ることができるのです。

さて、複雑なマップを作るにはそれなりに複雑な処理となりますが、ここでは移動方向は一方向のみという制限を付けることにします。

マップの基本パターンは他のパターンと同様に32×32ドットとすると、画面横方向には20個のパターンを並べることができますから、マップデータはパターンが横に20個並んでおり、それが縦方向に展開する構造とし、各パターンは番号で管理するものとします。ですからマップデータを管理する為の変数は、次のように単純な2次元配列とすればよいでしょう。

```
#define  PYMAX 240-1
#define  PXMAX 20-1
unsigned char map_data[PYMAX+1][PXMAX+1];
```

まず、マップエディタを作る前に、ここで定義したデータをディスクにセーブしたりロードする為の専用の関数を作ることにします。

LIST4-2.C

```
/*
    マップデータ用ファイル管理
*/

#include "list3-16.c"    /* list3-16.c のインクルード */
#define  PYMAX 240-1     /* マップデータY方向最大値 */
#define  PXMAX 20-1      /* マップデータX方向最大値 */

extern void vramdotset( int, int, char, int, int, int,
                        int, int, int, int, int );
```

```
/* マップデータ格納用変数配列の定義 */
unsigned char map_data[PYMAX+1][PXMAX+1];

/* マップデータをディスクへセーブする */
map_data_save(void) {
    char fname[64];
    FILE *sfp;
    char c;
    int i, j, k, l, fsin;

    vramdotset(0, 0, 0xff, 1, 2, 4, 8, 36, 400, 0, 0);
    fsin = 0;
    while(1) {
        i = 3;
        while(i) {
            printf("¥n¥ninput save file name¥n");
            i = getname(fname);
            if(i == 1) {
                fsin = 2;
                break;
            }
            if(i == 2) file_disp(fname);
            else i = 0;
        }
        if(fsin == 2) break;
        if(access(fname, 0) == 0) {
            printf( "¥n%s is allready exist ¥ncontinue (Y-N) ? ",
                    fname );
            while(1) {
                c = getch();
                if(c == 'n' || c == 'N') {
                    fsin = 2;
                    break;
                }
                if(c == 'Y' || c == 'y') {
                    printf("Y");
                    break;
                }
            }
            if(fsin == 2) break;
        }
        else  printf("save file name  = %s ", fname);
```

```
        sfp = fopen(fname, "wb");
        if(NULL == sfp) {
            fsin = 3;
            break;
        }
        fsin = 0;
        for(i = 0; i <= PYMAX; ++i) {
            for(j = 0; j <= PXMAX; ++j) {
                if(fputc(map_data[i][j], sfp) == EOF) {
                    fsin = 1;
                    break;
                }
            }
            if(fsin == 1) break;
        }
        fclose(sfp);
        break;
    }
    fmes(fsin, fname);
    return(fsin);
}
/* マップデータをディスクからロードする */
map_data_load(void)
{
    char fname[64];
    FILE *sfp;
    int c, i, j, k, l, s, done, fsin;
    struct ffblk ffblk;

    vramdotset(0, 0, 0xff, 1, 2, 4, 8, 36, 400, 0, 0);
    fsin = 0;
    while(1) {
        i = 3;
        while(i) {
            printf("¥n¥nﾏｯﾌﾟ ﾃﾞｰﾀ ﾛｰﾄﾞ");
            printf("¥ninput map data file name¥n");
            i = getname(fname);
            if(i == 1) {
                fsin = 2;
                break;
            }
```

```
                if(i == 2) file_disp(fname);
                else i = 0;
            }
            if(fsin == 2) break;
            printf("load file name  = %s ", fname);
            sfp = fopen(fname, "rb");
            if(NULL == sfp) {
                fsin = 3;
                break;
            }
            fsin = 0;
            for(i = 0; i <= PYMAX; ++i) {
                for(j = 0; j <= PXMAX; ++j) {
                    if((c = fgetc(sfp)) == EOF) {
                        fsin = 1;
                        break;
                    }
                    map_data[i][j] = c;
                }
                if(fsin == 1) break;
            }
            fsin = 0;
            fclose(sfp);
            break;
        }
        fmes(fsin, fname);
        return(fsin);
}
```

マップエディタというのは、基本的にマップデータ用に定義した配列へ、画面に展開したパターンに対するパターン番号を格納したり、逆に、配列に格納してあるパターン番号から、画面を復元する機能を持たせればよいわけです。では、プロジェクトLIST4-3.PRJまで進んでください。

LIST4-3.C

```
/*
        マップエディタ
*/
#include <graphics.h>
#include <conio.h>
```

```
#include <stdio.h>
#include <bios98.h>
#include <dir.h>
#include <io.h>

#define NPAT    80*32
#define RIGHT   '6'
#define LEFT    '4'
#define UP      '8'
#define DOWN    '2'
#define ESC     0x1b
#define SPACE   ' '
#define LOAD1   'l'
#define LOAD2   'L'
#define LOAD3   'ﾘ'
#define SAVE1   's'
#define SAVE2   'S'
#define SAVE3   'ﾄ'
#define XMAX    11-1
#define YMAX    4-1
#define XMAX2   20-1
#define YMAX2   6-1
#define XMIN    0
#define YMIN    0
#define XMIN2   0
#define YMIN2   0
#define PXMAX   20-1
#define PYMAX   240-1

extern char far *c_blue;
extern char far *c_red;
extern char far *c_green;
extern char far *c_itsty;
extern unsigned char map_data[PYMAX+1][PXMAX+1];

extern int  pattern_in(unsigned int, int);
extern int  pattern_out(unsigned int, int);
extern int  gload(void);
extern int  map_data_save(void);
extern int  map_data_load(void);

void right_set(void);
```

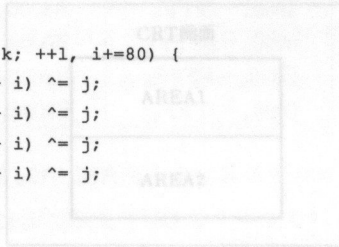
```
void left_set(void);
void down_set(void);
void up_set(void);
void boxcursol(int);
void disp_map(int);
int mwindow = 36,
    mpwindow = 0x7d00 - 80 * 32 * 6,
    box_inf = 1,
    x   = 0,
    y   = 0,
    x2  = 0,
    y2  = 5,
    px  = 0,
    py  = 0,
    py0 = 0,
    boxadr;

/* マップエディタ用メイン関数の定義 */
void main()
{
    int i, j, k, l, m, n, key;
    grphinit();
    pointer_set();
    printf("¥nﾊﾟﾀｰﾝ ﾃﾞｰﾀ ﾛｰﾄﾞ");
    if(gload() == 0) {
        for(k = m = n = 0, l = mwindow; k <= YMAX; ++k, l += NPAT) {
            for(i = 0, j = l; i < 11; ++i, j+=4) {
                n = pattern_in(m++, j);
                if(n != 0) break;
            }
            if(n != 0) break;
        }
        clrscr();
        key = 0;
        pattern_out(x + y * (XMAX + 1), 0);
        disp_map(0);
        boxcursol(y * 32 * 80 + x * 4 + mwindow);
        boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
        while(key != ESC) {
            key = getch();
            switch(key) {
                case RIGHT:
```

```
                right_set();
                break;
            case LEFT:
                left_set();
                break;
            case UP:
                up_set();
                break;
            case DOWN:
                down_set();
                break;
            case SPACE:
                boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
                pattern_out(x + y * (XMAX + 1),
                            y2 * 32 * 80 + x2 * 4 + mpwindow);
                map_data[py][px] = (char)(x + y * (XMAX + 1));
                boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
                break;
            case LOAD1:
            case LOAD2:
            case LOAD3:
                map_data_load();
                py0 = 0;
                boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
                disp_map(0);
                boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
                pattern_out(x + y * (XMAX + 1), 0);
                break;
            case SAVE1:
            case SAVE2:
            case SAVE3:
                map_data_save();
                py0 = 0;
                boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
                disp_map(0);
                boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
                pattern_out(x + y * (XMAX + 1), 0);
                break;
            default : break;
        }
    }
}
```

```
    else {
        printf("¥n press any key");
        getch();
    }
    closegraph();
}

/* ボックスカーソル右方向への移動 */
void right_set(void)
{
    KEY_INFO keyinf;
    keyinf.cmmd = 2;
    if(bios98key(&keyinf) != 0) {
        boxcursol(y * 32 * 80 + x * 4 + mwindow);
        x += 1;
        if(x > XMAX) {
            x = XMIN;
            y += 1;
            if(y > YMAX) y = YMAX;
        }
        boxcursol(y * 32 * 80 + x * 4 + mwindow);
        pattern_out(x + y * (XMAX + 1), 0);
    }
    else {
        boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
        ++x2;
        if(x2 > XMAX2) x2 = XMIN2;
        px = x2;
        boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
    }
}

/* ボックスカーソル左方向への移動 */
void left_set(void)
{
    KEY_INFO keyinf;
    keyinf.cmmd = 2;
    if(bios98key(&keyinf) != 0) {
        boxcursol(y * 32 * 80 + x * 4 + mwindow);
        x -= 1;
        if(x < XMIN) {
            x = XMAX;
```

```
            y -= 1;
            if(y < YMIN) y = YMIN;
        }
        boxcursol(y * 32 * 80 + x * 4 + mwindow);
        pattern_out(x + y * (XMAX + 1), 0);
    }
    else {
        boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
        --x2;
        if(x2 < XMIN2) x2 = XMAX2;
        px = x2;
        boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
    }
}

/* ボックスカーソル下方向への移動 */
void down_set(void)
{
    KEY_INFO keyinf;
    keyinf.cmmd = 2;
    if(bios98key(&keyinf) != 0) {
        boxcursol(y * 32 * 80 + x * 4 + mwindow);
        ++y;
        if(y > YMAX) y = YMIN;
        boxcursol(y * 32 * 80 + x * 4 + mwindow);
        pattern_out(x + y * (XMAX + 1), 0);
    }
    else {
        boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
        ++y2;
        if(y2 > YMAX2) {
            --y2;
            --py0;
            if(py0 < 0) py0 = 0;
            else disp_map(py0);
        }
        --py;
        if(py < 0) py = 0;
        boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
    }
}
```

```
/* ボックスカーソル上方向への移動 */
void up_set(void)
{
    KEY_INFO keyinf;
    keyinf.cmmd = 2;
    if(bios98key(&keyinf) != 0) {
        boxcursol(y * 32 * 80 + x * 4 + mwindow);
        --y;
        if(y < YMIN) y = YMAX;
        boxcursol(y * 32 * 80 + x * 4 + mwindow);
        pattern_out(x + y * (XMAX + 1), 0);
    }
    else {
        boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
        --y2;
        if(y2 < YMIN2) {
            ++y2;
            ++py0;
            if(py0 > PYMAX - 5) py0 = PYMAX - 5;
            else disp_map(py0);
        }
        ++py;
        if(py > PYMAX) py = PYMAX;
        boxcursol(y2 * 32 * 80 + x2 * 4 + mpwindow);
    }
}

/* マップの表示 */
void disp_map(int py0)
{
    int i, j, k, l;
    for(i = 0, k = 0x7d00 - 32 * 80; i < 6; ++i) {
        for(j = 0, l = k; j <= PXMAX; ++j, l+=4) {
            pattern_out((unsigned int)map_data[i+py0][j], l);
        }
        k -= 32 * 80;
    }
}


void xorvramdt(int, int, int);
```

```
/* ボックスカーソルの表示 */
void boxcursol(int k)
{
    int xmax = 32, ymax = 32;
    int i, j;
    if(box_inf == 1) {
        j = (ymax - 1) * 80;
        for(i = 0; i < xmax / 8; ++i, ++j) {
            xorvramdt(k + i, 0xff, 1);
            xorvramdt(k + j, 0xff, 1);
        }
        j = xmax / 8 + k + 79;
        k += 80;
        xorvramdt(k, 0x80, ymax - 2);
        xorvramdt(j, 0x01, ymax - 2);
    }
    else box_inf = 1;
}

/* 指定アドレスへ指定データを書き込む */
void xorvramdt(int i, int j, int k)
{
    int l;
    for(l = 0; l < k; ++l, i+=80) {
        *(c_blue  + i) ^= j;
        *(c_red   + i) ^= j;
        *(c_green + i) ^= j;
        *(c_itsty + i) ^= j;
    }
}
```

LIST4-3.PRJ
list2-0 list2-1 list3-0 list4-0 list4-2 list4-3

このマップエディタで注意しなければならないのは、マップデータの終わりの判断をどのようにするかです。マップデータの変数はunsigned

charとしましたから、格納できる数値は0〜255となります。そこで、本書では、この中の0番をマップデータの終わりを表す番号とすることにしました。ですから、パターン番号の0番はマップを構成するパターンとしては使えないということになります。なお、この関数の先頭では、マップデータを0で初期化をしています。

■ GDCによるスムーズスクロール

マップが完成したなら、次に画面のスクロールにチャレンジしてみようではありませんか。この画面スクロールはTVゲームに限らずグラフィックス処理には欠かせないテクニックであるといえます。さて、このスクロール処理には色々な方法がありますが、ここでは標準的なGDC（Graphic Display Controler）による方法を紹介しましょう。

GDCとは、PC-9801シリーズの初代から登載されているグラフィック画面コントロール専用のLSIです。このLSIが持っている機能には画面を分割して表示する機能があって、これをうまく利用することで画面のスクロールが可能となるのです。このスクロールはハード的なスクロールであることから、ハードウェアスクロールともいわれます。図4-2を参照してください。

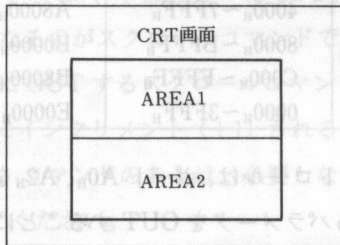



図4-2　GDCによる分割画面とCRT画面との対応

GDCではこのように任意のアドレス（ワード単位）でひとつの画面を二分割して表示することが可能なのです。この時、表示開始位置はAREA1の左上となります。また、このAREA1とAREA2の位置はワード単位で任意に設定できますから、図4-3のようにAREA1とAREA2の構成を連続的に変化させることによって、表示開始位置を徐々に移動することが可能となり、見かけ上スムーズなスクロールが実現できるのです。

さて、GDCのコントロールで分かりにくいのはアドレス管理の方法で

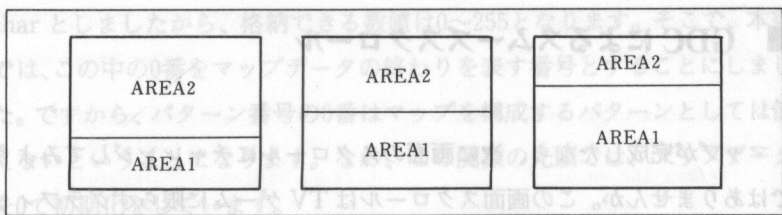



図4-3 GDCによる画面分割によるスクロール

しょう。GDCとCPUは同じG.V.RAMをアクセスするのですが、割り付けてあるアドレスはまったく異なります。しかも同じG.V.RAMであるにもかかわらずアクセス単位はCPUが8ビットであるのに対して、GDCは16ビットを単位としています。表4-1にそれぞれが管理しているアドレスを示しますから参考にしてください。

表4-1 GDCとCPUとのG.V.RAMの対応

	GDC	CPU
ブルー面	4000_H～$7FFF_H$	$A8000_H$～$AFFFF_H$
レッド面	8000_H～$BFFF_H$	$B0000_H$～$B7FFF_H$
グリーン面	$C000_H$～$FFFF_H$	$B8000_H$～$BFFFF_H$
輝度面	0000_H～$3FFF_H$	$E0000_H$～$E7FFF_H$

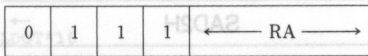
実際のGDCのコントロールは、ポート$A0_H$、$A2_H$を介してコマンドとそのコマンドに付随するパラメータをOUTすることによって行われます。この時、GDC内部のバッファの状態をチェックしなければならないことに注意してください。GDCでは処理を効率よく進めるために16個のFIFO(First In First Out)バッファがあって、コマンドやパラメータを一時的に蓄えて実行する構造となっているのですが、バッファがいっぱいであればウェイトを取らなければならないからです。GDCの状態はポートA0Hから表4-2のようなステータスフラグとして得られます。

バッファの空き状態はビットの2でチェックすることができます(b2=1)。また、バッファの状態はビットの1でもチェックできますが、この場合にはコマンドやパラメータを送る前に一回一回チェックしなければなりま

表4-2　GDCのステータスフラグ

ビット	内　容
0	DATA READY
1	FIFO BUFFER FULL
2	FIFO BUFFER EMPTY
3	DRAWING
4	DMA EXECUTE
5	VERTICAL SYNC
6	HORIZONTAL BLANK
7	LIGHT PEN DETECT

せん。さて、コマンドですが、スクロールコマンドでは次のような構造となります。

0	1	1	1	← RA →

このように上位ニブルは7、下位ニブルはGDC内部のRAMアドレス(以後RAと略記)となるのがスクロールコマンドです。RAの初期値は0となりますから、初めにOUTするスクロールコマンドは0x70となります。また、RAは自動的にインクリメント（+1）されるので、コマンドは一度送るだけですみます。コマンドのあとは、必要となるパラメータを送ります。図4-4を参照してください。

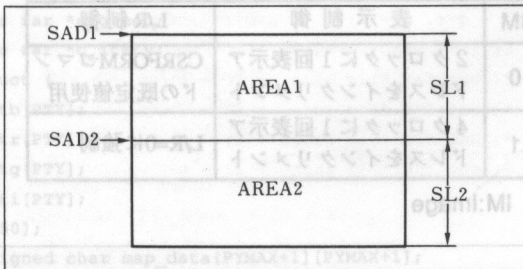



図4-4　表示画面との対応

このようにひとつの画面をAREA1とAREA2に二分割して、それぞれの開始アドレスとライン数をSAD1、SL1と、SAD2、SL2とすると、各パラメータは表4-3のような順番で送ることになります。

表4-3 スクロールコマンドの各パラメータ

内蔵RAMマップ（RA：GDC内部のRAMアドレス）

RA	7	6	5	4	3	2	1	0
0	← SAD1L →							
1	← SAD1H →							
2	← SL1L →				0	0	0	0
3	*	IM	← SL1H →					
4	← SAD2L →							
5	← SAD2H →							
6	← SL2L →				0	0	0	0
7	*	IM	← SL2H →					

*:DAD+2

DAD+2	機能
0	"1" によるインクリメント（DAD+1→DAD）
1	"2" によるインクリメント（DAD+2→DAD）

DAD:Display Address

IM	表示制御	L/R制御
0	2クロックに1回表示アドレスをインクリメント	CSRFORMコマンドの既定値使用
1	4クロックに1回表示アドレスをインクリメント	L/R=0に強制

IM:Image

ところで、最近の機種ではGDCの動作クロックが5MHzと2.5MHzの二通りがあり、ユーザーが選択して使用できるようになっていますから、

動作クロックについても考慮しなければなりません。どちらのクロックを使用するかはディップスイッチ2番の8によって選択します（OFF で2.5 MHz、ON で5MHz)。なお、変更に際してはシステムを再起動しなければなりません。

当然のことながらGDC のパラメータもこの動作クロックに合わせる必要があります。スクロールコマンドでは、2.5MHz で IM ビット＝0、5MHz でIM ビット＝1となります。では以上のことを踏まえて次のプログラムへと進んでください。

LIST4-4.C

```
/*
   GDCによるスムーズスクロール
*/
#define PTY    32
#define SCLDOT 2
#define SCLD16 SCLDOT*16
#define SCLPTC SCLDOT*40
#define GSTADR 0x4000
#define GENADR 0x7e80
#define STLINE 0x1900
#define ENLINE 0x1900-SCLD16
#define VEND   400*80
#define PYMAX  240-1
#define PXMAX  20-1

extern char far *c_blue;
extern char far *c_red;
extern char far *c_green;
extern char far *c_itsty;
extern struct {
   long dtb[PTY];
   long dtr[PTY];
   long dtg[PTY];
   long dti[PTY];
} pattern[50];
extern unsigned char map_data[PYMAX+1][PYMAX+1];

int enemy_table1[20];
int enemy_sign;
```

```
char gdchz = 0;

void scroll(int gvs, int gve, int sign)
{
    struct BYTE {
        char low;
        char high;
    };
    union DATA {
        short dataw;
        struct BYTE datab;
    };
    union DATA
        no1,
        no2,
        ad1,
        ad2;
    static int
        sclad1,
        sclno1,
        sclno2,
        map_ptn_y,
        map_y,
        map_x;
    int l, m, n, p, q;

    if(sign == 0) {                     /* sign=0ならば全画面をクリアする */
        map_ptn_y = PTY-SCLDOT;
        map_x  = 0;
        map_y  = 0;
        sclad1 = GSTADR + SCLPTC;
        for(l = gve, m = 0; m < VEND / 4; ++m, l+=4) {
            if(l >= VEND) l-=VEND;
            *(long far *)(c_blue   + l) = 0;
            *(long far *)(c_red    + l) = 0;
            *(long far *)(c_green + l) = 0;
            *(long far *)(c_itsty + l) = 0;
        }
    }
    else {                              /* sign=1ならば不要部分の消去をする */
        for(m = 0, p = gve; m < 20; ++m, p+=4) {
            for(l = 0, q = p; l < SCLDOT; ++l, q+=80) {
                *(long far *)(c_blue   + q) = 0;
```

```
                *(long far *)(c_red   + q) = 0;
                *(long far *)(c_green + q) = 0;
                *(long far *)(c_itsty + q) = 0;
            }
        }
    }
    while((inportb(0xa0) & 0x4) == 0);
    outportb(0x00a2, 0x70);             /* スクロールコマンドの送出 */
    sclad1 -= SCLPTC;
    if(sclad1 <= GSTADR) {
        sclad1 = GENADR;
        sclno1 = 0;
        sclno2 = STLINE;
        ad1.dataw = GSTADR;
        ad2.dataw = GENADR - SCLPTC;
        no1.dataw = ENLINE;
        no2.dataw = SCLD16;
    }
    else {
        sclno1 += SCLD16;
        sclno2 -= SCLD16;
        ad1.dataw = sclad1;
        ad2.dataw = GSTADR;
        no1.dataw = sclno1;
        no2.dataw = sclno2;
    }
    no1.datab.high |= gdchz;
    no2.datab.high |= gdchz;
    while((inportb(0xa0) & 0x20) != 0);
    while((inportb(0xa0) & 0x20) == 0);
    outportb(0xa0, ad1.datab.low); /* GDCへ各パラメータを送る */
    outportb(0xa0, ad1.datab.high);
    outportb(0xa0, no1.datab.low);
    outportb(0xa0, no1.datab.high);
    outportb(0xa0, ad2.datab.low);
    outportb(0xa0, ad2.datab.high);
    outportb(0xa0, no2.datab.low);
    outportb(0xa0, no2.datab.high);
    if(sign == 0) {             /* sign=0なら全画面を砂漠のデータで埋める */
        for(l = gvs; l < VEND; l+=2) {
            *(int far *)(c_blue  + l) = 0;
            *(int far *)(c_red   + l) = 0xaaaa;
            *(int far *)(c_green + l) = 0xaaaa;
            *(int far *)(c_itsty + l) = 0xaaaa;
        }
```

```
        }
        else {                              /* sign=1ならばマップデータを表示する */
            for(map_x = 0; map_x < 20; ++map_x, gvs+=4, gve+=4) {
                n = map_data[map_y][map_x];
                for( l = 0, m = map_ptn_y, p = gvs, q = gve;
                        l < SCLDOT; ++l, ++m, p += 80, q += 80 ) {
                    *(long far *)(c_blue  + p) = pattern[n].dtb[m];
                    *(long far *)(c_red   + p) = pattern[n].dtr[m];
                    *(long far *)(c_green + p) = pattern[n].dtg[m];
                    *(long far *)(c_itsty + p) = pattern[n].dti[m];
                }
            }
            map_ptn_y -= SCLDOT;
            if(map_ptn_y < 0) {
                map_ptn_y = PTY - SCLDOT;
                ++map_y;
                if(map_data[map_y][0] == 0 || map_y >= 240) map_y = 0;
                enemy_sign = gvs - 80 * 2;
                for(n = 0; n < 20; ++n) {
                    enemy_tablel[n] = map_data[map_y][n];
                }
            }
            else enemy_sign = 0;
        }
}
```

さて、LIST4-4.Cでは、スクロールと共にマップの表示処理も同時に行っています。マップの表示開始位置を示しているのが関数の引数gvsで、表示終了位置を示しているのがgveです。すなわち、gvsの位置でスクロールした分だけマップを表示し、gveの位置で不要部分の消去をするわけです。

引数のsignは画面の初期状態を表示するためのサインを意味しています。sign＝0で初期画面の表示及び使用変数の初期化作業をしています。では、この関数を実際に動かしてみましょう。main()関数とプロジェクトファイルは次のようになります。なお、プログラムの先頭では使用するパターンを格納してあるファイルとマップデータを格納してあるファイルをロードする構造としました。

LIST4-5.C

```
/*
    スクロールのテスト
*/
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <bios98.h>
#include <dos.h>

#define PTY         32
#define NPAT        80*PTY
#define MAKE_ESC    0
#define SCLDOT      2
#define PTY0        42*80*8
#define VSTADR      16
#define MAKE_RETURN 0x1c

extern int  pattern_in(unsigned int, int);
extern int  pattern_out(unsigned int, int);
extern int  gload(void);
extern int  map_data_load(void);
extern void scroll(int, int, int);

extern char gdchz;

int  mwindow = 36;

void main()
{
    int i, j, k, l, m, n, py0, pye, keydat;
    KEY_INFO keyinf;

    if(inportb(0x31) & 0x80) gdchz = 0;
    else                     gdchz = 0x40;
    grphinit();
    pointer_set();
    printf("¥nﾊﾟﾀｰﾝ ﾃﾞｰﾀ ﾛｰﾄﾞ");
    if(gload() == 0) {
        for(k = m = n = 0, l = mwindow; k < 4; ++k, l+=NPAT) {
            for(i = 0, j = l; i < 11; ++i, j+=4) {
                n = pattern_in(m++, j);
```

```
                if(n != 0) break;
            }
            if(n != 0) break;
        }
        map_data_load();
        clrscr();
        keyinf.cmmd = 1;
        keydat = 0;
        py0 = 80 * VSTADR;
        pye = 400 * 80 - 80 * SCLDOT;
        scroll(80 * VSTADR, py0, 0);
        while(keydat == 0) {
            keydat = bios98key(&keyinf);
            scroll(py0, pye, 1);
            py0 -= 80 * SCLDOT;
            if(py0 < 0) py0 = 80 * 400 - 80 * SCLDOT;
            pye -= 80*SCLDOT;
            if(pye < 0) pye = 80 * 400 - 80 * SCLDOT;
        }
    }
    else {
        printf("¥n press any key");
        getch();
    }
    closegraph();
}
```

LIST4-5.PRJ

```
list2-0
list2-1
list3-0
list4-0
list4-2
list4-4
list4-5
```

■ ヒーロー登場

いかがでしたか、実際にデザインしたマップがスクロールしていく過程を見るのは一種感動するものがあるのではないでしょうか。さて、いよいよ作り出した新しい世界の中を動き回るヒーローの登場と願いましょう。

あるパターンをグラフィック画面上に表示する場合、パターンの余白から見えるはずの背景をいかに表示するかという重ね合わせ処理が問題となります。

これを完全に処理するためには、どの部分がパターンで、どこからが余白であるのかを表す専用のデータが必要となります。データ構造としてはパターンがある所は0、無い所は1として背景との論理演算ANDを実行することで背景をパターンの形でくり抜くという操作をするわけです。後は、くり抜いた背景の所に論理演算のORでパターンを表示すればよいのです。

実は、このように言葉で説明するのは簡単なのですが、これをプログラムするとなると、かなり複雑になってしまいます。そこで本書では、この重ね合わせの問題を解決するために、砂漠(口絵6)以外には移動できないという制限を付けることにします。すなわち、パターンの余白の部分にあらかじめ背景の砂漠のパターンで埋めておくわけです。このような制限を付けることによってプログラムは飛躍的に簡単になります。しかも、パターンの移動後に残る残骸を元の背景に復元する処理も単純に砂漠を描けばよいわけですから、さらに処理が簡単になるわけです。もっとも、マップを描く時には必ず砂漠が存在しないと移動できないということになりますから注意してください。では、LIST4-6.Cへと進んでください。

LIST4-6.C

```
#define PTY          32
#define NPAT         80*PTY
#define MAKE_RIGHT   0x48
```

```
#define MAKE_LEFT   0x46
#define BREAK_RIGHT 0xc8
#define BREAK_LEFT  0xc6
#define XMAX        76
#define XMIN        0
#define ON          1
#define OFF         0
#define CRASH       1
#define CONTINUE    0
#define UE          1
#define MIGI        2
#define HIDARI      3
#define VEND        400*80

extern char far *c_blue;
extern char far *c_red;
extern char far *c_green;
extern char far *c_itsty;

extern int keydat;
extern struct {
    long  dtb[PTY];
    long  dtr[PTY];
    long  dtg[PTY];
    long  dti[PTY];
} pattern[50];

int check(unsigned int, int, int);

int px = 38;

/* 車の表示及び移動処理 */
int car_disp(int py, int keep_py)
{
    static int
        dir_m = OFF,
        dir_h = OFF,
        pn  = 0;
    unsigned int j, k;
    int dir, keep_px, r;

    dir = UE;
    keep_px = px;
```

```
    switch(keydat) {
        case BREAK_RIGHT:
            dir_m = OFF;
            if(dir_h == ON) keydat = MAKE_LEFT;
            else break;
        case MAKE_LEFT:
            dir = HIDARI;
            dir_h = ON;
            for(k = 0, j = px + 3 + py; k < PTY; j+=80, ++k) {
                if(j >= VEND) j -= VEND;
                *(c_blue  + j) = 0;
                *(c_red   + j) = 0xaa;
                *(c_green + j) = 0xaa;
                *(c_itsty + j) = 0;
            }
            --px;
            if(px < XMIN) {
                dir = UE;
                px  = XMIN;
            }
            break;
        case BREAK_LEFT:
            dir_h = OFF;
            if(dir_m == ON) keydat = MAKE_RIGHT;
            else break;
        case MAKE_RIGHT:
            dir = MIGI;
            dir_m = ON;
            for(k = 0, j = px + py; k < PTY; j+=80, ++k) {
                if(j >= VEND) j -= VEND;
                *(c_blue  + j) = 0;
                *(c_red   + j) = 0xaa;
                *(c_green + j) = 0xaa;
                *(c_itsty + j) = 0;
            }
            ++px;
            if(px > XMAX) {
                dir = UE;
                px  = XMAX;
            }
            break;
        default :
            dir_m = OFF;
            dir_h = OFF;
            break;
```

```
    }
    for(k = 0, j = keep_px + NPAT + py; k < 8; j+=80, ++k) {
        if(j >= VEND) j -= VEND;
        *(long far *)(c_blue + j) = 0;
        *(long far *)(c_red   + j) = 0xaaaaaaaa;
        *(long far *)(c_green + j) = 0xaaaaaaaa;
        *(long far *)(c_itsty + j) = 0;
    }
    if(check(px + py, keep_px + keep_py, dir) == CONTINUE) {
        switch(pn) {
            case  0:
                for(k = 0, j = px + py; k < PTY; j+=80, ++k) {
                    if(j >= VEND) j -= VEND;
                    *(long far *)(c_blue  + j) = pattern[0].dtb[k];
                    *(long far *)(c_red   + j) = pattern[0].dtr[k];
                    *(long far *)(c_green + j) = pattern[0].dtg[k];
                    *(long far *)(c_itsty + j) = pattern[0].dti[k];
                }
                break;
            case  1:
                for(k = 0, j = px + py; k < PTY; j+=80, ++k) {
                    if(j >= VEND) j -= VEND;
                    *(long far *)(c_blue  + j) = pattern[1].dtb[k];
                    *(long far *)(c_red   + j) = pattern[1].dtr[k];
                    *(long far *)(c_green + j) = pattern[1].dtg[k];
                    *(long far *)(c_itsty + j) = pattern[1].dti[k];
                }
                break;
            case  2:
                for(k = 0, j = px + py; k < PTY; j+=80, ++k) {
                    if(j >= VEND) j -= VEND;
                    *(long far *)(c_blue  + j) = pattern[2].dtb[k];
                    *(long far *)(c_red   + j) = pattern[2].dtr[k];
                    *(long far *)(c_green + j) = pattern[2].dtg[k];
                    *(long far *)(c_itsty + j) = pattern[2].dti[k];
                }
                break;
            default : break;
        }
        ++pn;
        if(pn > 2) pn = 0;
        r = 0;
    }
    else {
        px = keep_px;
```

```
        r = 1;
        dir_m = OFF;
        dir_h = OFF;
        keydat= 1;
    }
    return(r);
}

/* 移動方向が移動可能かを判断する */
int check(unsigned int pxy, int keep_pxy, int dir)
{

    unsigned char a, b, c;
    unsigned int pxy2;
    int  i, j, k, l, r;
    long o, p, q;

    pxy2 = pxy;
    r = CONTINUE;
    switch(dir) {
        case MIGI:
            pxy += 3;
        case HIDARI:
                a = *(c_blue  + pxy);
                b = *(c_red   + pxy);
                c = *(c_green + pxy);
                if(a == 0 && b == 0xaa && c == 0xaa) {
                    pxy += 80 * 16;
                    if(pxy >= VEND) pxy -= VEND;
                    a = *(c_blue  + pxy);
                    b = *(c_red   + pxy);
                    c = *(c_green + pxy);
                    if(a == 0 && b == 0xaa && c == 0xaa) {
                        pxy += 80*16;
                        if(pxy > VEND) pxy -= VEND;
                        a = *(c_blue  + pxy);
                        b = *(c_red   + pxy);
                        c = *(c_green + pxy);
                        if(a == 0 && b == 0xaa && c == 0xaa);
                        else r = CRASH;
                    }
                    else r = CRASH;
                }
                else r = CRASH;
        case UE:
```

```
                o = *(long far *)(c_blue  + pxy2);
                p = *(long far *)(c_red   + pxy2);
                q = *(long far *)(c_green + pxy2);
                if(o == 0 && p == 0xaaaaaaaa && q == 0xaaaaaaaa);
                else r = CRASH;
                break;
            default : break;
        }
        if(r == CRASH) {
            for(i = 0; i < 3; ++i) {
                for(l = 3; l <= 5; ++l) {
                    printf("¥a");
                    while((inportb(0xa0) & 0x20) != 0);
                    while((inportb(0xa0) & 0x20) == 0);
                        for(k = 0, j = keep_pxy; k < PTY; j+=80, ++k) {
                        if(j >= VEND) j -= VEND;
                        *(long far *)(c_blue  + j) = pattern[l].dtb[k];
                        *(long far *)(c_red   + j) = pattern[l].dtr[k];
                        *(long far *)(c_green + j) = pattern[l].dtg[k];
                        *(long far *)(c_itsty + j) = pattern[l].dti[k];
                    }
                }
            }
            for(k = 0, j = keep_pxy; k < PTY; j+=80, ++k) {
                if(j >= VEND) j -= VEND;
                *(long far *)(c_blue  + j) = 0;
                *(long far *)(c_red   + j) = 0xaaaaaaaa;
                *(long far *)(c_green + j) = 0xaaaaaaaa;
                *(long far *)(c_itsty + j) = 0;
            }
        }
        return(r);
}
```

さて、ここでの処理のポイントは、ヒーローとなる車をデザインしたパターン番号の0〜2を順番に表示する処理の所でswitch〜case文を使用していることです。これは、構造体を参照する場合のインデックスをダイレクトに数値を使って指定する為の工夫なのですが、この方が、変数による場合よりも若干スピードが早いということです。

また、移動可能か否かのチェックをG.V.RAMから直接データを読んで判断しています。砂漠のデータは、B面が0で、R面とG面のデータが

0xaaaa……と並んでいる構造となりますから、車の進行方向と左右でG.V.RAMの数値をチェックするわけです。なお、左右でチェックする場合には、パターンの上中下の3箇所でチェックをします。では実際に動かしてみましょう。なお、LIST4-7.Cはキーボード設定用となりますから注意してください。

LIST4-7.C

```
#include <dos.h>
#include <bios98.h>
#define STATUS_PORT     0x43
#define COMMAND_PORT    0x43
#define DATA_PORT       0x41
#define EOI             0
#define COMMAND         0x16
#define EOIDATA         0x20
#define ERRCODE         0x38

int keydat,
    random= 3751,
    seed1 = 3751,
    seed2 = 1357;

void frandom(void);

/* キーボード割り込み関数の定義 */
void interrupt keyset(void)
{
    int x;
    frandom();
    x = inportb(STATUS_PORT) & ERRCODE;
    if(x == 0) {
        outportb(COMMAND_PORT, COMMAND);
        keydat = inportb(DATA_PORT);
    }
    outportb(EOI, EOIDATA);
}

/* 疑似乱数用関数 */
void frandom(void)
{
    seed1  = random;
    random <<= 2;
```

```
    random += seed1;
    ++random;
    random = random & 0xf;
    random +=2;
}

void interrupt (*keepvector9)();
void interrupt keyset(void);

/* キーボード割り込みの初期設定 */
void initkey(void)
{
    keepvector9 = getvect(9);
    setvect(9, keyset);
}

/* キーボード割り込みを復元する */
void termkey(void)
{
    KEY_INFO keyinf;
    setvect(9, keepvector9);
    keyinf.cmmd = 3;
    bios98key(&keyinf);
}
```

LIST4-8.C

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>

#define PTY           32
#define NPAT          80*PTY
#define MAKE_ESC      0
#define SCLDOT        2
#define PTY0          42*80*8
#define VSTADR        16
#define MAKE_RETURN 0x1c

extern int keydat, px;
extern int  pattern_in(unsigned int, int);
extern int  pattern_out(unsigned int, int);
extern int  gload(void);
extern void initkey(void);
```

```
extern void termkey(void);
extern int  car_disp(int, int);
extern void scroll(int, int, int);
extern int  map_data_load(void);

extern char gdchz;

int  mwindow = 36;

void main()
{
    int i, j, k, l, m, n, py, py0, pye, r;
    unsigned int kyori;

    if(inportb(0x31) & 0x80) gdchz = 0;
    else                     gdchz = 0x40;
    grphinit();
    pointer_set();
    clrscr();
    printf("¥nﾊﾟﾀｰﾝ ﾃﾞｰﾀ ﾛｰﾄﾞ");
    if(gload() == 0) {
        for(k = m = n = 0, l = mwindow; k < 4; ++k, l += NPAT) {
            for(i = 0, j = l; i < 11; ++i, j += 4) {
                n = pattern_in(m++, j);
                if(n != 0) break;
            }
            if(n != 0) break;
        }
        map_data_load();
        clrscr();
        initkey();
        gotoxy(1, 1);
        printf("ＲＡＬＬＹ　９８");
        keydat = 1;
        py0 = 80 * VSTADR;
        while(keydat != MAKE_ESC) {
            scroll(80 * VSTADR, py0, 0);
            i  = PTY0;
            py = PTY0;
            px = 38;
            py0 = 80 * VSTADR;
            pye = 400 * 80 - 80 * SCLDOT;
            keydat = 0x70;
```

```
            kyori   = 0x0;
            r = 0;
            gotoxy(68, 1);
            printf("距離");
            while(keydat != MAKE_ESC && r == 0) {
                gotoxy(73, 1);
                printf("%u m     ", kyori);
                ++kyori;
                r = car_disp(py, i);
                while(keydat == 0x70);
                scroll(py0, pye, 1);
                i = py;
                py -= 80 * SCLDOT;
                if(py < 0)  py  = 80 * 400 - 80 * SCLDOT;
                py0 -= 80 * SCLDOT;
                if(py0 < 0) py0 = 80 * 400 - 80 * SCLDOT;
                pye -= 80 * SCLDOT;
                if(pye < 0) pye = 80 * 400 - 80 * SCLDOT;
            }
        }
        termkey();
    }
    else {
        printf("¥n press any key");
        getch();
    }
    closegraph();
}
```

LIST4-8.PRJ

```
list2-0
list2-1
list3-0
list4-0
list4-2
list4-4
list4-6
list4-7
list4-8
```

■ 敵の出現

敵、すなわち、進行を邪魔する存在ですが、本書では砂漠の中から突然、回転しながら現れる三角形の板としました。そうです、アニメーションのサンプルで使ったあのパターンです。アニメーションといってもパターン番号を順番に変化させるという簡単なものです。ここで特に難しいのは出現させるタイミングでしょう。進行を妨げるのですから、移動不可能な砂漠以外の所に出現したのではあまりにも芸がないからです。次のプログラムでは、疑似乱数を使ってマップデータから砂漠のパターンをサーチして、敵を起動するタイミングとしてあります。では、一気に最後のプロジェクトファイルであるLIST4-10.PRJまで進んでください。

LIST4-9.C

```
#define PTY     32              /* パターンを構成するライン数 */
#define SCLDOT 2                /* スクロールドット数 */
#define VEND    400*80          /* G.V.RAMのENDアドレス */
#define SABAKU 6                /* 砂漠パターン番号 */
#define STAPTN 7                /* 敵パターンアニメーションスタート番号 */
#define ENDPTN 18               /* 敵パターンアニメーションエンド番号    */
extern char far *c_blue;        /* B面へのポインタ */
extern char far *c_red;         /* R面へのポインタ */
extern char far *c_green;       /* G面へのポインタ */
extern char far *c_itsty;       /* I面へのポインタ */
extern int enemy_table1[20];    /* 敵X座標決定用 */
extern int enemy_sign;          /* 敵出力サイン */
extern int random;              /* 乱数格納用 */
extern int px;                  /* パターンX座標 */
/* パターン格納用構造体 */
extern struct {
    long dtb[PTY];
    long dtr[PTY];
    long dtg[PTY];
    long dti[PTY];
} pattern[50];
```

```
/* 敵の状態管理用構造体の定義 */
struct {
    int inf;
    int x;
    int y;
    int dy;
    int ptn;
} enemy[4];
void frandom(void);

/* 敵の表示用関数の定義 */
void enemy_disp(int enemy_set)
{
    int i, j, k, l, m, n;
    if(enemy_set == 1) {
        for(i = 0; i < 4; ++i) {
            /* 敵の状態別処理 */
            switch(enemy[i].inf) {
                case 0:
                    /* 敵表示開始処理 */
                    if(enemy_sign != 0) {
                        frandom();
                        for(j = 0, k = random; j < 20; ++j) {
                            ++k;
                            if(k >= 20) k = 0;
                            j = enemy_table1[k];
                            if(j == SABAKU) {
                                enemy[i].inf = 1;
                                k = k + k + k + k;
                                enemy[i].x   = k;
                                enemy[i].y   = 16;
                                enemy[i].dy  = enemy_sign+k;
                                enemy[i].ptn = STAPTN * 4;
                                enemy_sign   = 0;
                                break;
                            }
                        }
                    }
                    break;
                case 1:
                    /* 敵Y座標更新 */
                    enemy[i].y += SCLDOT;
                    if(enemy[i].y > 400-10*PTY) enemy[i].inf = 2;
```

```
                break;
            case 2:
                /* 敵パターン更新処理 */
                n = enemy[i].ptn;
                n >>= 2;
                if(n >= ENDPTN) enemy[i].inf = 0;
                for(j = 0, m = enemy[i].dy, k = PTY-1;
                    j < PTY; ++j, --k) {
                    *(long far *)(c_blue   + m)
                            = pattern[n].dtb[k];
                    *(long far *)(c_red    + m)
                            = pattern[n].dtr[k];
                    *(long far *)(c_green + m)
                            = pattern[n].dtg[k];
                    *(long far *)(c_itsty + m)
                            = pattern[n].dti[k];
                    m -= 80;
                    if(m < 0) m += VEND;
                }
                ++ enemy[i].ptn;
                break;
            default : break;
        }
    }
}
else {
    /* 敵情報テーブル初期化 */
    for(i = 0; i < 4; ++i) enemy[i].inf = 0;
    enemy_sign = 0;
}
}
```

LIST4-10.C

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>

#define PTY         32
#define NPAT        80*PTY
#define MAKE_ESC    0
#define SCLDOT      2
```

```
#define PTY0        42*80*8
#define VSTADR      16
#define MAKE_RETURN 0x1c

extern int keydat, px;
extern int  pattern_in(unsigned int, int);
extern int  pattern_out(unsigned int, int);
extern int  gload(void);
extern void initkey(void);
extern void termkey(void);
extern int  car_disp(int, int);
extern void scroll(int, int, int);
extern int  map_data_load(void);
extern void enemy_disp(int);
extern void frandom(void);

extern char gdchz;

int  mwindow = 36;

void main()
{
    int i, j, k, l, m, n, py, py0, pye, r;
    unsigned int kyori;

    if(inportb(0x31) & 0x80) gdchz = 0;
    else                     gdchz = 0x40;
    grphinit();
    pointer_set();
    clrscr();
    printf("¥nﾊﾟﾀｰﾝ ﾃﾞｰﾀ ﾛｰﾄﾞ");
    if(gload() == 0) {
        for(k = m = n = 0, l = mwindow; k < 4; ++k, l += NPAT) {
            for(i = 0, j = l; i < 11; ++i, j+=4) {
                n = pattern_in(m++, j);
                if(n != 0) break;
            }
            if(n != 0) break;
        }
        map_data_load();
        clrscr();
        initkey();
        gotoxy(1, 1);
```

```
        printf("R A L L Y   9 8");
        keydat = 1;
        py0 = 80 * VSTADR;
        while(keydat != MAKE_ESC) {
            enemy_disp(0);
            scroll(80 * VSTADR, py0, 0);
            i   = PTY0;
            py  = PTY0;
            px  = 38;
            py0 = 80 * VSTADR;
            pye = 400 * 80 - 80 * SCLDOT;
            keydat = 0x70;
            kyori  = 0x0;
            r = 0;
            gotoxy(68, 1);
            printf("距離");
            while(keydat != MAKE_ESC && r == 0) {
                gotoxy(73, 1);
                printf("%u m    ", kyori);
                ++kyori;
                r = car_disp(py, i);
                while(keydat == 0x70)    frandom();
                enemy_disp(1);
                scroll(py0, pye, 1);
                i = py;
                py -= 80 * SCLDOT;
                if(py < 0)  py  = 80 * 400 - 80 * SCLDOT;
                py0 -= 80 * SCLDOT;
                if(py0 < 0) py0 = 80 * 400 - 80 * SCLDOT;
                pye -= 80 * SCLDOT;
                if(pye < 0) pye = 80 * 400 - 80 * SCLDOT;
            }
        }
        termkey();
    }
    else {
        printf("¥n press any key");
        getch();
    }
    closegraph();
}
```

LIST4-10.PRJ

```
list2-0
list2-1
list3-0
list4-0
list4-2
list4-4
list4-6
list4-7
list4-9
list4-10
```

実行してみると、たった一種類の敵ですが今までにない雰囲気となったはずです。ここでは一種類の敵だけとしましたが、さらに多くの敵を出現させ、攻撃パターンに工夫を凝らしたりすれば、このゲームもより面白いものになると思います。

なお、機種によっては実行スピードが遅く感じられる場合があるかと思いますが、その時には、MS-DOSのコマンドラインから直接起動してみてください。

さて、サンプルプログラムということで、背景描写や敵のパターンも含めて33種類のパターンだけでゲームを作ってみました。市販のゲームと比べると今一歩の感は否めないようですが、それもそのはずです。市販のゲームでは、数百ものパターンを用意するのが普通です。しかも、プログラム、パターンデザイン、グラフィックデザイン、シナリオ、音楽というように、それぞれの分野のプロが専門分野を担当するというのが一般的ですから、無理もないことなのです。

とはいいながら、アイデアしだいでは世界中に受け入れられるゲームが生まれる可能性がありますから、後は読者の創造性にまかせることとして本書でのゲーム作りの幕としましょう。

■ まとめ

C言語によるゲーム作りはいかがでしたか、このようにグラフィックス処理も遊び感覚でできるゲームから入ると楽しく進められたのではないでしょうか。プログラミング作業も視覚で確かめながら進めていくことができ、しかも、ゲームは様々なテクニックが要求されるために、プログラムテクニックの修練の場としては最適といえるのではないでしょうか。

グラフィックス処理には様々な分野がありますが、基本はG.V.RAMへのポインタをどのように扱うかに尽きると思います。CAD、CAMに代表されるCGの世界も、ファミコンに代表されるTVゲームの世界も、すべてG.V.RAMをどのように扱うのかという問題へと行き着いてしまうのです。

PC-9801シリーズは、CAD、CAMやTVゲーム用の専用機ではありませんが、グラフィックス処理専用のハードウェアであるGDC (Graphic Display Controller) やGRCG (Graphic Charger)、EGC (Enhanced Graphic Charger)などが登載されているために、かなりの処理が可能になります。これらのハードウェアのコントロールも、本書程度のG.V.RAMの処理を把握していないと、やはり困難であるといえます。

ところで、このGDCに関しては、それだけで一冊の本ができるぐらいの内容がありますので、本書ではGDCの持っている機能の中でもスクロール機能だけに絞り、他の機能を解説することはしませんでしたが、その他様々な機能を応用すれば、たとえば直線描画機能を光線銃の軌跡に利用するなど、さらにおもしろいものができると思います。

GRCGやEGCに関しては、登載機種が限られているために、サンプルプログラムでの使用を避けました。しかし、特にEGCには、シフト機能を有した4画面同時転送機能や、ビット演算機能、さらにEGCとGDCとを組み合わせて使うなど、ゲーム制作にはうってつけの機能が豊富にあります。登載されている機種であれば、これらの機能を使わないのは、まさに「宝

の持ち腐れ」といったところではないでしょうか。たとえば、パターンの完全な重ね合わせ処理や、左右のドット単位スクロール処理、多重スクロールなどアイデアしだいで応用範囲はかなり広いものがありますから、ぜひ、EGCの活用も研究してみて下さい。

また、TURBO CではTURBOアセンブラと組み合わせることによって、Cのプログラム中にインラインアセンブラとしてアセンブリ言語のプログラムを書くことができます。もし、TURBOアセンブラがなくても、MASMで開発したマシン語とのリンクが簡単にできるために、EGCなどの機能が登載されていない機種を使用している方や、汎用性の観点からPC-9801シリーズ固有のGDC、GRCG、EGCというハードウェアは使いたくないならば、C言語とマシン語のリンクという方法を検討してみるのもよいでしょう。もちろんこれらの登載機種であっても、たとえばパターン表示関数などのような、特にスピードを要求される一部の処理をマシン語に置き換えるというのは、度々用いられている手法です。

本書をステップにGDCやEGCを駆使した本格的なグラフィックス処理にチャレンジしてみてください。もしも、マシン語やゲームに興味がある方であれば、『はじめてのマシン語』(啓学出版)、『マシン語ゲームプログラミング』(アスキー)『マシン語ゲームグラフィックス』(小学館)などにも取り組んでみてください。また、最後になりましたが、本書および続編に対する御意見、御希望などありましたら聞かせていただければ幸いです。

索引

[プログラムソースのディスクサービスのお知らせ]

本書に掲載したプログラムのソースコードとパターンデータの入ったディスクを販売いたします。ご希望の方は巻末に添付してある振り替え用紙に住所、氏名、電話番号、希望のディスクメディアをご記入の上、お申し込み下さい。

価格：3000円（送料込み）

・本ディスクにはTURBO CおよびBG1ファイル、MS-DOSシステムは含まれていません。

・本ディスク中のプログラムをそのまま、あるいは改変して使用した結果として生じた損害について、著作者および出版者はいっさいの責任を負いません。

・プログラムからCopyright(c) Manabu Aoyamaの表記を削除しないで下さい。

[著者紹介]

青山　学（あおやま まなぶ）

1958年　東京生まれ。

青山学院大学理工学部卒業。コンピュータエンジニアを経て、現在はゲームデザイナー。

大型コンピュータからパソコンまで、言語、機種にこだわらないのを信条としている。著書として、『PC-9801シリーズはじめてのマシン語』『8086マシン語秘伝の書』(以上啓学出版)、『PC-9801シリーズマシン語ゲームプログラミング』(アスキー)、『マシン語ゲームグラフィックス』(小学館)などがある。

趣味は、スキー、テニスなど。

PC-9801＋TURBO Cグラフィックスに強くなる本　ゲーム編

1991年9月30日　第1刷発行

著　者　青（あお）　山（やま）　学（まなぶ）

発行所　啓学出版株式会社

代表者　三井数美

郵便番号　101

東京都千代田区神田神保町1-46

電　話　東京03(3233)3795[販売部]

東京03(3233)3731[編集部]

振　替　東京3-109286

印刷／昭和工業写真印刷所

製本／徳　住　製　本　所

ISBN4-7665-0900-5　　本書の定価はカバーに表示してあります。

Printed in Japan　　*SHI*

通常払込料金
加入者負担

払込通知票

各票の※印欄は、払込人において記載してください。

口座番号	東京 3 109286	金額	※ 億 千 百 十 万 千 百 十 円
加入者名	啓学出版株式会社	料金	払込み　特殊　円
払込人住所氏名	※（郵便番号　　）	備考	
		受付局日附印	

記載事項を訂正した場合は、その箇所に訂正印を押してください。

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）

切り取らないで郵便局にお出しください。

通常払込料金
加入者負担

払込票

口座番号	東京 3 109286
加入者名	啓学出版株式会社
金額	※ 億 千 百 十 万 千 百 十 円
払込人住所氏名	※
備考	受付局日附印

このたびは、小社の通信販売をご利用いただき、有難うございます。

○この払込票は、本領収書となります。
○送金手数料は、弊社負担です。
○この払込票は、書籍のご注文にもご利用いただけます。

啓学出版株式会社
〒101 東京都千代田区神田神保町1-46
☎ 03—3233—3795(代)
FAX 03—3233—3730

—払込内訳—

品名	単価	数量	金額
グラフィックスに強くなる本 ゲーム編	3,000(税込)		
(注) メディアの種別を明記してください。(対応機種PC-9800シリーズ) 3.5インチ2HD・5インチ2HD			
合計			

通信欄	追加注文・連絡事項等、御記入ください。

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。 (郵政省)

PC-9801+TURBO C
グラフィックスに強くなる本
●青山学著
ゲーム編